Central Processor

1.1 Introduction

The Central Processor (CP) controls the high-speed I/O devices and the main memory of the Dandelion. It provides short-latency memory access and ALU service for the integral I/O controllers and can emulate the Mesa Processor as defined by the Mesa Processor Principles of Operation. It is composed of about 160 standard chips and resides entirely on one 11" by 17" printed circuit card located in slot 3.

This chapter presents at the hardware structures of the Central Processor and its interfaces with the rest of the Dandelion. Another manual, the *Dandelion Microcode Reference (DMR)*, presents the assembler microcode format and is intermixed with hardware details and examples.[1]

The CP is a microprogrammed, 16-bit general-purpose computer. The microcode control store can hold up to 4096 48-bit microinstructions[2] and can be read or written by the low-speed Input/Output Processor (IOP). Each microinstruction is decoded and executed in 137 nanoseconds, a *cycle*.[3] All microinstruction operations are completed in one cycle; instruction execution is not pipelined over several cycles, except that while one is being executed its successor is being read from the microstore.

Cycles are grouped into *clicks*, where one click equals three successive cycles labeled c1, c2, and c3. Cycles are always enumerated in order c1, c2, c3, and then c1 again.[4] This sequence is never interrupted or altered; accordingly, both targets of a two-way branch must be specified with the same cycle number. (Strictly speaking, this is necessary only if the target microinstructions contain cycle-dependent operations.) The microcoder's task of aligning instructions so that they execute in successive cycles is a necessary outcome of the fixed-tasking, click structure. Moreover, when one desires code which is speed optimized, this structure usually requires the elimination of three microinstructions instead of one.

While the three microinstructions of a click are executing, a memory read or write can be performed: the address is sent to the memory in c1, a single data word may be sent during c2, and data is returned from memory in c3. A memory operation can *only* be initiated in cycle 1.

Clicks are grouped into *rounds*: five successive clicks (numbered 0..4) comprise a round, which is two microseconds in duration. Each click of a round is permanently allocated to one or more of the I/O controllers. If an I/O controller does not request the service of its correspondent *task* microcode, the Emulator-microcode task runs during that click instead of the device-microcode task. When there is a transition between tasks, the hardware preserves the outgoing task's microprogram counter and restores when it runs again.

The click is a basic microcode time unit: devices and the Emulator are serviced in units of clicks and the microcode can transfer exactly one memory word in this time. Since a click is 411 nanoseconds in duration, the maximum bandwidth available through a CP's click is 7.8 mbits/s.

The CP is implemented using four 2901 bit-slice chips plus external memories and registers. The 2901 provides 17 registers readily accessible to the microcoder, the usual logical and arithmetic functions, and single bit shifting.

Available to the microprogrammer and external to the 2901 are four register sets (U, RH, IB, and Link), a four-bit rotator, the I/O registers and memory, and four Emulator registers (stackP, ibPtr, pc16, and MInt). There are no task specific registers: all registers can be addressed by all tasks.

1.2 Microinstruction Format

The microinstruction format strikes a balance between some naturally opposing structures: control store width versus control store size, encoding schemes versus decoding hardware constraints, and coverage of all possible data operations versus exclusion of impracticable operations. The format was designed with the goal that frequently applied operations are encoded in the least number of bits. Furthermore, it was designed so that the most important Mesa Emulator and I/O operations execute in one click. The format is illustrated and summarized in Figure 1.

A 48-bit microinstruction has three major parts: 2901-control bits, miscellaneous functions, and a "goto"-address field. The field names are abbreviated as:

rA, rB	R registers A and B
aS, aF, aD	ALU source address, function, destination address
ep	even parity
Cin	2901 carry input
enSU	enable stack/U registers
mem	memory operation
fS	function fields selector
fX, fY, fZ	function fields X, Y, and Z
INIA	intermediate next instruction address.

The 2901-control bits occupy the first word: rA, rB, aS, aF, and aD. The "goto" address, INIA, utilizes 12 bits. INIA is a control-store-destination address unless condition bits, specified by the previous microinstruction, are *or'd* into it, resulting in a branch or dispatch. Thus, every microinstruction is a potential jump instruction.

The fS field is broken into two subfields: fS[0-1] and fS[2-3]. These control the deciphering of the fY and fZ fields, respectively. Both the fY and fZ fields have four possible enumerations as defined by fS:

The fY field can, depending on fS[0-1]: (1) name a branch or multi-way dispatch, (2) specify a miscellaneous function, (3) name an I/O register to be loaded, or (4) equal the high nibble of an 8-bit constant. These four functions are called DispBr, fYNorm, IOOut, and Byte.

The fZ field can (1) enumerate a miscellaneous function, (2) equal a 4-bit constant, (3) be the low half of a U register address, or (4) name an I/O register to be read. These four classes are abbreviated fYNorm, Nibble, Uaddr, and IOXIn, respectively.

rA	rB	aS	aF	aD	ep	Cin	enSU	mem	fS	fX	fY	fZ	INIA
0	4	8	11	14	16				20	24	28	32	36 … 47

Field	Description
rA	2901 A reg addr, U addr [0-3]
rB	2901 B reg addr, RH addr
aS	2901 alu Source operand pair
aF	2901 alu Function
aD	2901 alu Destination/shift control
ep	Even Parity
Cin	2901 Carry In, Shift Ends, writeSU (if enSU = 1)
enSU	enable SU reg file
mem	MAR← (if c1), MDR← (if c2), ←MD (if c3)
fS	Function field Selector
fX	X Function
fY	Y Function
fZ	Z Function
INIA	Next Instruction Address

aS	R, S
0	A, Q
1	A, B
2	0, Q
3	0, B
4	0, A
5	D, A
6	D, Q
7	D, 0

aF	F
0	R + S + Cin
1	S − R − Cin'
2	R − S − Cin'
3	R or S
4	R and S
5	~R and S
6	R xor S
7	~R xor S

sh,,aD	B[rB]←	Q←	Ybus←
0	no write	F	F
1	no write	no write	F
2	F	no write	A
3	F	no write	F
4	F/2	Q/2	F
5	F/2	no write	F
6	2F	2Q	F
7	2F	no write	F

sh ← (fX = shift) OR (fX = cycle) OR (fY = cycle)

fS[0-1]	fY =
0	DispBr
1	fYNorm
2	IOOut
3	Byte

fS[2-3]	fZ =	SU addr[0-7]
0	fZNorm	0,,stackP
1	Nibble	0,,stackP
2	Uaddr[4-7]	rA,,fZ | rA,,Y[12-15]* IF fZ = AltUaddr*
3	IOXIn	rA,,fZ | rA,,Y[12-15]* IF fZ = AltUaddr*

* as executed by previous u-instr

fX	fXNorm
0	pCall/Ret0
1	pCall/Ret1
2	pCall/Ret2
3	pCall/Ret3
4	pCall/Ret4
5	pCall/Ret5
6	pCall/Ret6
7	pCall/Ret7
8	Noop
9	RH←
A	shift
B	cycle
C	Cin←pc16
D	Map←
E	pop
F	push

fY	fYNorm	DispBr	IOOut
0	ExitKern	NegBr	IOPOData←
1	EnterKernel	ZeroBr	IOPCtl←
2	ClrIntErr	NZeroBr	KOData←
3	IBDisp	MesaIntBr	KCtl←
4	MesaIntRq	PgCarryBr	EOData←
5	stackP←	CarryBr	EICtl←
6	IB←	XRefBr	DCtlFifo←
7	cycle	NibCarryBr	DCtl←
8	Noop	XDisp	DBorder←
9	Map←	YDisp	PCtl←
A	Refresh	XC2npcDisp	MCtl←
B	push	YIODisp	←TStatus
C	ClrDPRq	XwdDisp	EOCtl←
D		XHDisp	KCmd←
E	ClrRefRq	XLDisp	←TIData
F	ClrKFlags	PgCrOvDisp	POData←

fZ	fZNorm	IOXIn
0	Refresh	←EIData
1	IBPtr←1	←EStatus
2	IBPtr←0	←KIData
3	Cin←pc16	←KStatus
4	Bank←	KStrobe
5	pop	←MStatus
6	push	←KTest
7	AltUaddr	EStrobe
8	Noop	←IOPIData
9		←IOPStatus
A		←ErrnIBnStkp
B		←RH
C	LRot0	←ibNA
D	LRot12	←ib
E	LRot8	←ibLow
F	LRot4	←ibHigh

pCall when NIA[7] = 0. pRet when NIA[7] = 1.

Equivalent names: XDirtyDisp = XLDisp; EtherDisp = YIODisp; TAddr← = ClrDPRq; TCtl← = PCtl←; TOData← = POData←

Figure 1. Dandelion CP Microinstruction Format

1.3 Registers and Data Paths

Figure 2 illustrates the registers and data paths layout for the CP. The area inside the dashed lines represents the internal components of the 2901 ALU. The Y bus corresponds to the Y output of the 2901 and the X bus is connected to the 2901 D input. Both the X and Y buses are available on the backplane.

1.3.1 R & Q Registers and 2901 Data Paths

Referring to Figure 2, there is a 16-word, two-port register file called the R registers. One of the output ports is labeled A and the other B. These are the "fast" registers of the CP and can be used to hold temporaries, memory data and addresses, and arithmetic operands.

Every cycle, the contents of the R register given by the register-A (rA) field of the microinstruction is available at the A port, and likewise for the B port. If rA = rB, then the same data appears at both ports.

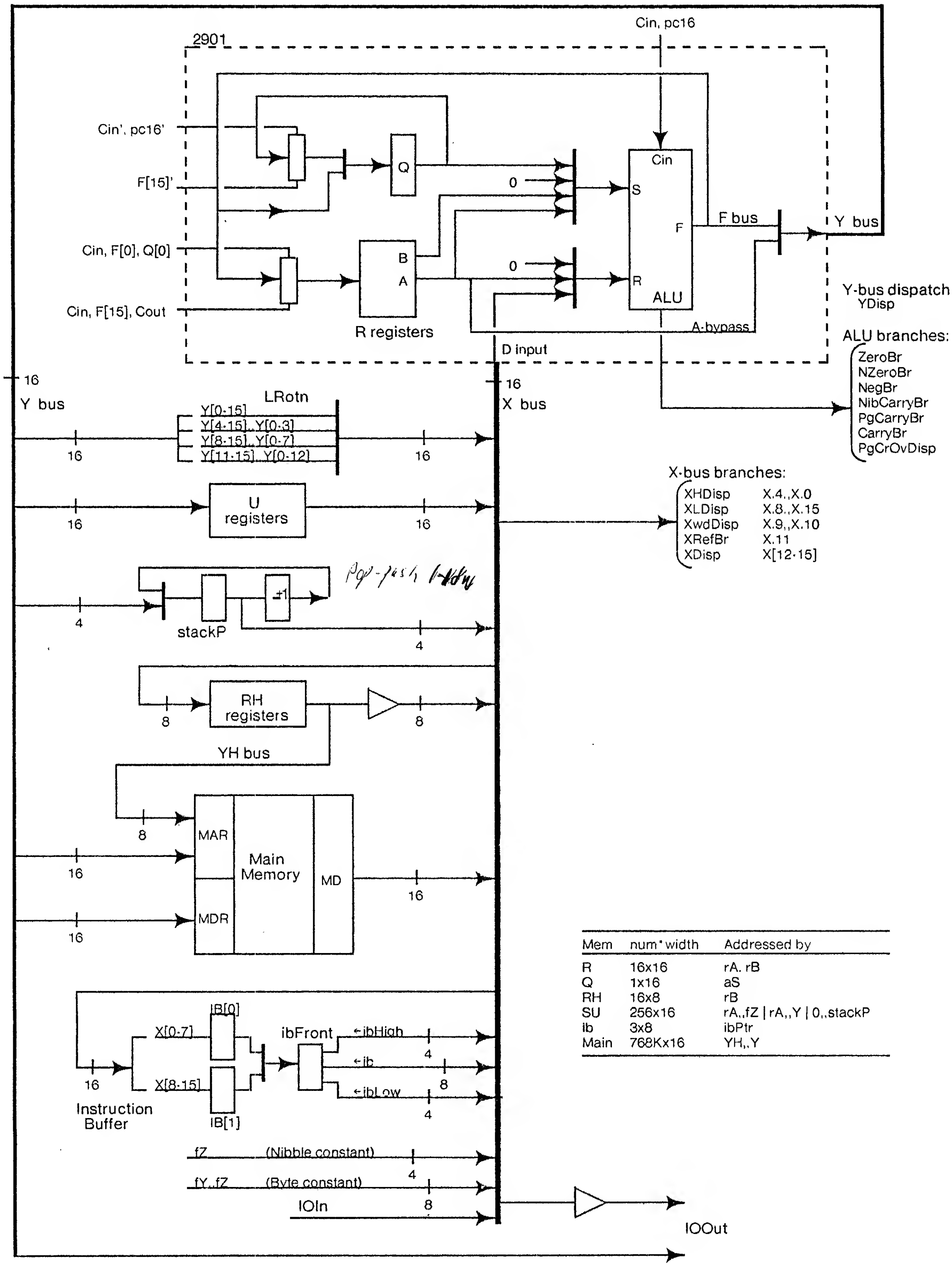
If the alu-Destination (aD) field specifies a write back into an R register, the rB field specifies which one: at the end of the cycle, register B is written with the ALU output (named F) or it is written with F shifted one bit.

The Q register holds 16 bits which can be written with the ALU output or its old value single-bit shifted left or right. It is implicitly referenced by the aS field of the microinstruction and can be used for double-word shifting.

The 2901 arithmetic unit has three inputs: R, S and Carryin (Cin). The R input can be set to the output of the A port, the value of the X bus, or zero. The S input can be driven by the output of the A or B ports, the value of the Q register, or zero. Cin can be either 0 or 1, or the value of the single-bit Emulator register pc16.

The 2901 can perform three arithmetic and five logical operations as specified by the alu-Function (aF) field. Arithmetic follows the two's-complement conventions. Three of the logical operations are symmetrical with respect to R and S: logical *or*, *and*, and *xor*. The remaining two logical operations complement R: ~R *xor* S and ~R *and* S.

Figure 3 shows a matrix of ALU computations as a function of possible aS and aF values. From the table it is clear there are many possible ways to generate zero within the ALU. All one's (0FFFF) is easily produced for some functions if rA = rB.



Mem	num*width	Addressed by
R	16x16	rA, rB
Q	1x16	aS
RH	16x8	rB
SU	256x16	rA,,fZ | rA,,Y | 0,,stackP
ib	3x8	ibPtr
Main	768Kx16	YH,,Y

Figure 2. Dandelion CP Data Paths

aF	Cin \ aS	(A,Q)	(A,B)	(0,Q)	(0,B)	(0,A)	(D,A)	(D,Q)	(D,0)	rA = rB = R (A,B)
R + S	0	A + Q	A + B	Q	B	A	X + A	X + Q	X	2R
	1	A + Q + 1	A + B + 1	Q + 1	B + 1	A + 1	X + A + 1	X + Q + 1	X + 1	2R + 1
S-R	0	Q-A-1	B-A-1	Q-1	B-1	A-1	A-X-1	Q-X-1	-X-1	-1
	1	Q-A	B-A	Q	B	A	A-X	Q-X	-X	0
R-S	0	A-Q-1	A-B-1	-Q-1	-B-1	-A-1	X-A-1	X-Q-1	X-1	-1
	1	A-Q	B-A	-Q	-B	-A	X-A	X-Q	X	0
R or S		A or Q	A or B	Q	B	A	X or A	X or Q	X	R
R and S		A and Q	A and B	0	0	0	X and A	X and Q	0	R
~R and S		~A and Q	~A and B	Q	B	A	~X and A	~X and Q	0	0
R xor S		A xor Q	A xor B	Q	B	A	X xor A	X xor Q	X	0
~R xor S		~A xor Q A xor ~Q	~A xor B A xor ~B	~Q	~B	~A	~X xor A X xor ~A	~X xor Q X xor ~Q	~X	-1

Figure 3. ALU Operations as a function of aS, aF, and Cin.

The F output of the ALU can be written into an R register, loaded into the Q register, or placed onto the Y bus. Although the F output is normally placed onto the Y bus, it is possible to route output-port A of the R register file onto the Y bus. This mode is called A-bypass or "A-pass-around."

The two-bit alu-Destination (aD) field, in combination with a one-bit value called sh, specifies whether R or Q are written and whether F or A-bypass is placed on the Y bus. The sh field is defined by certain functions of the microinstruction word (see Figure 1 for sh's definition). In general, when sh = 1 the F output is shifted one bit position before being written back into R or Q. This is accomplished inside the 2901 by 3-input multiplexers at the inputs to R and Q. What is shifted into the ends of R or Q determines the type of shift.

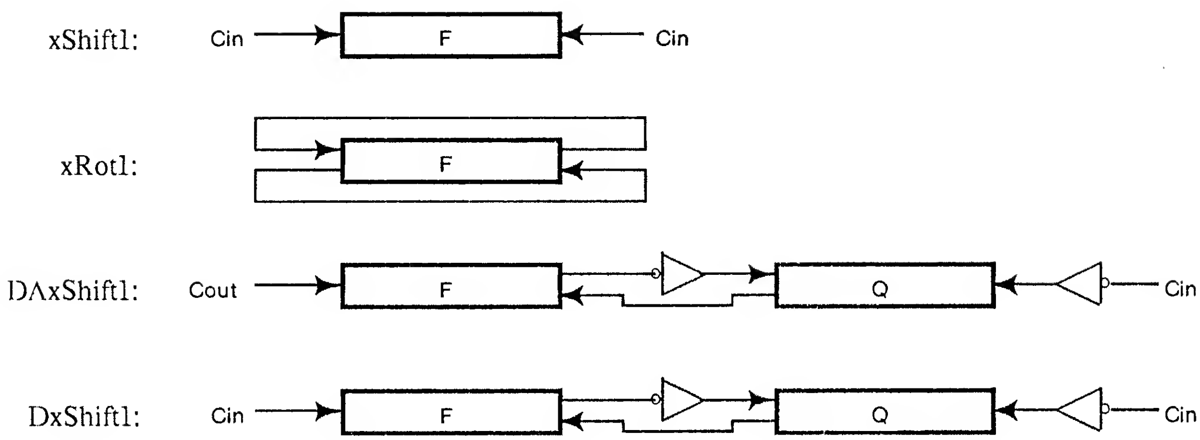
When sh concatenated with aD (sh,,aD) equals 001, neither an R register nor Q are written. This may be desired when writing an external register or when comparing two quantities. When sh,,aD = 000, Q is loaded with the ALU output. When equal to 010 or 011, an R register is loaded with the ALU output. The Y bus gets the ALU output in all cases except when sh,,aD = 010, where it receives the A-bypass value. Two general rules: When A-bypass is utilized an R register must be written and it is not possible to simultaneously write R and Q with F.

When sh = 1, a single-bit shifting operation is performed on the ALU output and/or Q. There are two major types of shift operations (Figure 4): a double-word shift of F,,Q and a single-word shift of F alone. These two types of shifting, combined with the two directions, are named by the four values of aD when sh = 1.

For single-word shifts, the Q register is unaffected and the R register gets twice or half of the ALU output. The end of F which is vacated by the shift operation is replaced by Cin or the bit shifted out of the opposite side of F (a single bit cycle).

For double-word shifts, both the ALU output and the Q register are shifted together. The low-order bit of the ALU output is "connected" with the high-order Q bit to form a 32-bit quantity. The high-order bit of F which is vacated by a right double shift can be written with Cin or the Carryout (Cout) of the current ALU computation. Similarly, the low end of Q is written with the complement of Cin (~Cin) if the shift direction is left. Note that the high bit of Q is written with the complement of the low bit of F. A general rule: Shift inputs into Q are *complemented*.

In summary, the following 2901 related restrictions apply: (1) When A-bypass is utilized an R register must be written, (2) it is not possible to simultaneously load R and Q, and (3) A-bypass cannot be used with single bit shifts or when loading Q.



function	aD	fX or fY
RShift1	1	shift
LShift1	3	shift
RRot1	1	cycle
LRot1	3	cycle
DARShift1	0	shift
DALShift1	2	shift
DLShift1	0	cycle
DRShift1	2	cycle

Figure 4. CP Single-Bit Shifting

1.3.2 External 2901 Data Paths

There are two major 16-bit data buses external to the 2901: the X bus and Y bus. Both are present on the backplane; however, they are *not* general purpose, bidirectional buses. The YH bus, an 8-bit extension of the Y bus, is used for memory addressing.

The Y bus is driven only by the Y output of the 2901. It can be used to supply a memory address, memory data, U register data, or device output data.

The X bus is the major system bus and is connected to multiple drivers and multiple receivers.[5] X bus sinks are: the D input of the 2901, the RH registers, the Instruction Buffer (IB), and controller output registers. X bus sources are: the U registers, RH registers, the IB, constants, memory data, and controller input registers. The IB, RH, and controller output registers receive data from the X bus so that they can be loaded directly from memory in one cycle.

Data can be passed from the Y bus to the X bus via a 4-bit rotator, called LRotn. Data can be rotated zero, four, eight, or twelve positions to the left, as specified by the fZ field. A zero rotation allows Y bus data to be placed unaffected onto the X bus; for example, when loading controller output registers from the ALU output.

Eight- or four-bit constants can be placed onto the X bus directly from the fY and/or fZ fields. The upper 8 or 12 bits of the X bus are set to zero.

The following table lists the registers which are addressable by the CP and which buses they are attached to:

Register	inputs from	Register	outputs to	
MAR←	YH,,Y	←MD	X	Memory
Map←	YH,,Y			
IB←	X	←ib, ←ibNA	X	Instruction Buffer
		←ibLow, ←ibHigh		X[12-15]
		~ibPtr	X[10-11]	
RH←	X[8-15]	←RH	X[8-15]	
U←	Y	←U	X	
stackP←	Y[12-15]	~stackP	X[12-15]	
MDR←	Y	EKErr	X[8-9]	
MCtl←	Y	←MStatus	X	Memory
KOData←	X	←KIData	X	Rigid Disk
EOData←	X	←EIData	X	Ethernet
POData←/TOData←	X	←TIData	X	LSEP/MagTape
IOPOData←	X	←IOPIData	X	IOP
KCtl←	X	←KStatus	X	Rigid Disk
KCmd←	X	←KTest	X	Rigid Disk
EICtl←	X	←EStatus	X	Ethernet
EOCtl←	X			
IOPCtl←	X	←IOPStatus	X	IOP
DCtl←	X			Display
DBorder←	Y			
DCtlFifo←	Y			
PCtl←/TCtl←	X	←TStatus	X	LSEP/MagTape
TAddr←	X			

1.3.3 U Registers

A 256-word register file, called the U registers, can be written from the Y bus and read onto the X bus. These 16-bit general purpose, "slow" registers are used to hold a 16-word stack, virtual page addresses, temporaries, counters, and constants.

With respect to accessibility, U registers are situated between main memory and the R registers: they cannot be both read and written in the same cycle, nor can they be used as an operand or destination register in 16-bit ALU arithmetic.

As illustrated below, there are three ways to form an 8-bit U register address: normal, stack-pointer, and alternate.

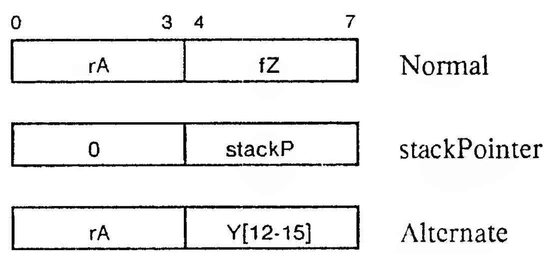


Figure 5. U Register Addressing Modes

In the normal mode, true when fS[2] = 1, the U register address is defined by the concatenation of the rA and fZ microinstruction fields. This sharing of the rA field between R and U register addresses has several implications. In general, a U register can be loaded into any R register since the rB field defines the write address. However, an arbitrary U register and an arbitrary R register cannot both be ALU operands unless the upper four bits of the U register address equal the R register address. This addressing mechanism partitions the U registers into sixteen 16-word banks where, in one cycle, a bank's U register can only be combined with the bank's corresponding R register.

In the stack-pointer addressing mode, used when fS[2] = 0, the U register is selected by the 4-bit stackPointer register (stackP) from the low bank; that is, the address is 0,,stackP. The stackP is *not* explicitly modified with this addressing mode and if the microinstruction also executes a pop or push function, the premodified stackP is used to access the U register.

The alternate mode provides indirect addressing and is used when fS[2] = 1 and fZ = AltUaddr for the *previously* executed microinstruction. In this mode, the low nibble of the U address equals the least significant Y bus nibble for the *previously* executed microinstruction--the same one that did the AltUaddr. Thus, instead of rA,,fZ, the U address is rA,,Y[12-15].

While reading or writing U registers, the fZ field can specify both a U register address and another function. Specifically, when fS[2-3] = 3, fZ can take on IOXIn values. This is commonly used to read an RH register or the IB while simultaneously writing a U register. When the stackPointer addressing mode is used, the fZ field is free to be interpreted as either fZNorm or a Nibble.

The U registers are also controlled by two other microinstruction fields: enSU and Cin. The enSU bit is 1 for any cycle which either reads or writes a U register. Cin must be 1 if written, and 0 if read. Thus, if a U register is written and the ALU function is addition or subtraction, these computations execute with Cin = 1. Note that normal two's complement subtraction implies Cin = 1.

1.3.4 RH Registers

Located on the X bus is the 16 by 8-bit RH register file, an extension of the R registers. The principle application of this small memory is to hold the highest-order memory address bits. Moreover, it can be utilized as general-purpose storage: flags, counters, temporaries, and subroutine return pointers (see *DMR*).

The RH registers are addressed by the rB field, and, since this field names the R register to be written, an RH register can only be written into its corresponding R register (or the Q register).

Like the U registers, they cannot be both read and written in the same cycle. An RH register is written from the low byte of the X bus when fX = RH← and is read onto X[8-15] when fZ = ←RH. Whenever it is read onto the X bus, the high half of the bus is set to zero.

Every cycle, the 8-bit YH bus is driven with the value of the addressed RH register, thereby supplying the high order memory address bits to the Memory Control card. However, these bits are only used by the memory if a MAR← or Map← is specified. As a corollary to the rule that RH registers cannot be simultaneously read and written, an RH register cannot be loaded if the microinstruction also executes a MAR← or Map←.

1.3.5 Instruction Buffer

The Instruction Buffer (IB) was designed to hold up to three Emulator macroinstructions or data bytes. It is used in a first-in, first-out manner. Data loaded into the IB from the X bus can be read back onto the X bus or be used to define a 256-way dispatch in control store. The IB is loaded by special Emulator "refill" microcode (sec. 1.6.4) while the actual control of the registers is accomplished by a hardware state machine.

The IB is maintained by the Emulator in a way that guarantees all macroinstructions will find necessary code segment operands there. Furthermore, the IB is where the 256-way dispatch is made on the next macroinstruction to be executed. This dispatch (IBDisp) occurs in c2 so that the next macroinstruction begins in c1, thereby adjoining the previous one. However, when IBDisp is executed and the buffer is not full, a microcode trap occurs and the refill microcode loads the buffer with more bytes from memory. If an IBDisp is executed and there is a pending interrupt (MInt = 1), special interrupt trap (IB-Refill) microcode runs instead of the refill microcode. Since the IB is so small, IBDisp's frequently trap; however, since the IB-Refill trap runs at memory speed, this scheme of supplying operand bytes to the macroinstructions is very efficient.

This scheme is efficient from both memory bandwidth and page-fault handling perspectives. In the former case, macroinstructions would otherwise have to call an operand-fetching subroutine, which would waste time becoming cycle aligned. In the latter case, macroinstructions need not worry about a page fault from the code segment. (The occurrence of a code segment page fault can add major complications to the implementation of macroinstructions since the microcode must, before processing the fault, restore the Mesa machine state to its value at the beginning of the instruction.) The IB insures that macroinstructions can always find code segment arguments present in the IB. In this sense, the IB is more like an operand data buffer than an instruction buffer.

The minimum number of bytes in the buffer required to prevent a IB-Refill trap is three (the maximum size of a Mesa macroinstruction) and they only occur between the execution of macroinstructions. The refill code completes in one click if the buffer requires two bytes and in two clicks for four. Because the buffer is small, the only codebytes which do not result in an IB-Refill trap are single-byte opcodes executed from even memory locations.

The instruction buffer itself consists of three 8-bit registers, called IB[0], IB[1], and ibFront. IB[0] holds the even code segment byte and IB[1] the odd. The bytes are shuffled through ibFront in even/odd, sequential order. There are four states which enumerate the location of data bytes among the holding registers. These states are indicated by the 2-bit register, ibPtr, and are defined below. The following diagram shows the four IB states (the cross-hatching indicates the position of the data bytes):

state name	bytes in IB	ibPtr
full	3	2
word	2	3
byte	1	1
empty	0	0

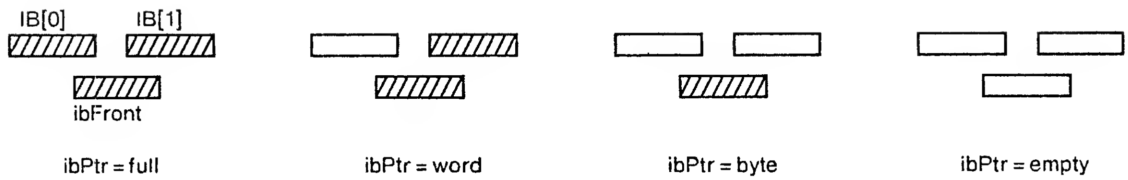


Figure 6. Instruction Buffer States

There are a total of 8 microinstruction functions which effect the IB. In general, the functions maintain the original even/odd byte ordering while updating ibPtr and ibFront. The following table lists the functions and their effect on ibPtr, ibFront, and the X bus. A discussion of the table follows, except that IB dispatches and IB-Refill traps are presented in sections 1.5.2 & 1.5.5.1.

function	new ibPtr	new ibFront	X bus ←
←ib	ibPtr-1	IF ibPtr[1] = 0 THEN IB[0] ELSE IB[1]	0,,ibFront
←ibNA	unchanged	unchanged	0,,ibFront
←ibHigh	unchanged	unchanged	0,,ibFront[0-3]
←ibLow	unchanged	unchanged	0,,ibFront[4-7]
IBDisp	ibPtr-1	IB[ibPtr[1]]	unaffected
AlwaysIBDisp	ibPtr-1	IB[ibPtr[1]]	unaffected
IB←	IF empty THEN word ELSE full	IF ibPtr = empty THEN X[0-7] ELSE unchanged	unaffected
IB←, IBPtr←1	IF empty THEN byte ELSE full	IF ibPtr = empty THEN X[8-15] ELSE unchanged	unaffected
IBPtr←0	word	IB[0]	unaffected
IBPtr←1	byte	IB[1]	unaffected
←ErrnIBnStkp	unchanged	unchanged	X[10-11]←~ibPtr

Figure 7. Effects of IB-related Functions

The IB is loaded from the X bus: the high-order, even byte is written into IB[0] and the low-order, odd byte into IB[1]. If the buffer is empty, then the X bus byte passes through IB[0] or IB[1] and is loaded directly into ibFront in one cycle; thus, the data can be used immediately in the cycle following the IB load.

The default IB write operation is that ibFront is written with X[0-7]. However, if IBPtr←1 is coincident with IB←, then ibFront is written with X[8-15] instead, thereby throwing away the even data byte. If there are one or two bytes in the buffer, then IB[0] and IB[1] are loaded and there is no feed through into ibFront.

ibFront can be read onto the X bus: when the microcoder specifies an ←ib or ←ibNA, ibFront is placed onto X[8-15] and the high byte of the X bus is set to zero.

There are several variations to this basic read. With the ←ibHigh function, ibFront[0-3] is placed onto X[12-15]. Analogously, ←ibLow places ibFront[4-7] onto X[12-15]. In both cases the upper 12 bits of the X bus are set to zero.

When ←ib is executed, a funneling process occurs: ibFront is loaded with the next byte from either IB[0] or IB[1] and ibPtr is "decremented" by one. ibPtr is gray code decremented: 2, 3, 1, and then 0. Thus, the low order bit of ibPtr divides the values of ibPtr into two classes with respect to refill: empty and not empty. (This scheme equates the empty and full states, but note that the buffer is not full when the IB-Refill trap occurs.)

Several of the microcode functions have no effect on the state of the buffer: The ←ibNA function (used to read the IB without advancing ibPtr), ←ibHigh, and ←ibLow do no change ibPtr. Also, like the RH and U registers, it is not possible to read and write IB simultaneously; hence, the combination of IB← and ←ib in the same cycle does nothing.

The functions IBPtr←0 and IBPtr←1, when autonomously used, merely load ibFront from IB[0] or IB[1], respectively. They typically occur in the cycle after the IB has been loaded with a jump-target codebyte, thereby selecting the even or odd destination opcode.

The complement of ibPtr can be read onto X[12-13] with the ←ErrnIBnStkp function.

1.3.6 stackP Register

The 4-bit stack pointer, stackP, is used to address one location from U register bank 0 (Sec. 1.3.3) and can be incremented or decremented independently of the 2901. The pop function decrements and the push function increments the stackP at the end of a cycle, performed modulo 16. Unlike the U and RH registers, the stackP can be read and written in the same cycle.

The stackP can be loaded from Y[12-15] with an fY function. However, one cycle must intercede between a stackP← and a microinstruction which uses the stack-pointer addressing mode and expects the new value. A pop or push can be used in the intervening instruction and appropriately modifies the value loaded.

The pop and push functions have been sprinkled throughout the microinstruction function fields to ameliorate the checking of stack overflow or underflow. The push function occurs in all three function fields while pop is in fX and fZ. An outcome of this arrangement is that when push is specified in the same microinstruction as pop, the stackP does not change: it does not matter how many pop's or push's there are, as long as there are both, the stackP is unaffected. Also, multiple pops or pushs in the same instruction do not decrement or increment the stackP by more than one. Multiple pop and push functions are used to check for stack overflow or underflow (sec. 1.5.5.2).

1.3.7 pc16 Register

The pc16 register is designed to serve as a low-order, 1-bit extension of an R register; namely, the R register which holds the Emulator's macroprogram counter (PC). That is, pc16 can be used as the byte index of a PC memory address.

If fX or fZ is Cin←pc16, the pc16 bit becomes the carry input of the 2901 and pc16 is inverted at the conclusion of the cycle. Thus, Cin←pc16, in combination with ALU addition and subtraction, properly adjusts the 17-bit byte program counter PC,,pc16 (See *DMR*).

Since Cin is also the shift ends (Sec. 1.3.1), Cin←pc16 can be used to shift pc16 into the low-order bit of an R register in one cycle, thereby reconstructing a word program counter in an R register.

Due to the hardware implementation of the carry input, when the Cin field of the microinstruction is 0, the fX version of Cin←pc16 must be used. If Cin = 1, then either the fX or fZ version of Cin←pc16 can be specified.

1.4 Main Memory Interface

This section discusses the interface between the CP and the memory system. As outlined earlier, a memory address is sent to the Memory Controller in c1, any data to be written is sent during c2, and returning data is available in c3. Every click is a potential memory operation: if the Emulator kept the memory 100% busy and there were no I/O, it would have available up to 2.4 megawords/s (38 mbits/s) of bandwidth.

The memory system accepts two types of addresses: real or virtual. Real references result in a read or write to the addressed location itself. Virtual references cause the memory system to ignore the low byte of the address and then, using the remaining 16 bits, read or write the Map, located at real address 10000 hex.

For both reference types, when the mem field is set in c2 a write occurs (MDR←) and when set in c3 a read occurs (←MD). If both a read and write are specified in the same click, the original value is returned and then the location is overwritten. Furthermore, if a click specifies a MDR← or ←MD without a corresponding MAR← then memory is not written and a potential memory Error trap does not occur.

As outlined in section x.xx, the memory system is available in a variety of sizes: real address size from 192K to 768K words and virtual address size from 4 to 16 megawords. This section assumes the maximum of both ranges: 20-bit real addresses and 24-bit virtual addresses.

1.4.1 Real Address References

When the mem bit is true in cycle 1, a real reference is caused. The microcoder specifies a real reference by using the MAR← macro in c1. The memory address is sent to the Memory Control card on the YH and Y buses. The Y bus can be driven from either the 2901's F bus or A-bypass; hence addresses can be either pre or postmodified. The YH bus, which supplies the high-order address bits, is always driven by the RH register addressed by rB. Furthermore, YH[0-3] are ignored by the memory.

Several important things happen with a MAR←: the 2901 is divided such that the high half executes a fixed function, a special "address-overflow" branch is enabled, and an MDR← or IBDisp in the next cycle is canceled if the branch is taken. Moreover, if a MAR← is executed with YH[4-7] = 0 and the display controller is enabled and actually transferring bits to the monitor, then the click is suspended (See sec. 1.5.6.5).

MAR← Effect: Split 2901

If mem = 1 in c1, the 2901 is divided such that the high half executes with its aS and aF inputs equal to (0,B) and (aF *or* 3), while the low half executes the aS and aF values given by the microinstruction. This causes the high byte of the ALU output to equal the high byte of the R register addressed by rB (or its complement if aF is in [4..7]). Thus, assuming the Y bus is driven from the F bus, the 20-bit real address is rhB[4-7],,rB[0-7],,F[8-15].

However, if A-bypass is specified, the lowest 16 address bits come from the R register addressed by rA. Hence, the 20-bit real address is rhB[4-7],,rA[0-15].

An outcome of this bipartition is that a carry out from the low half does not propagate into the high half: the high byte of rB remains unchanged after a MAR← (unless aF is in [4..7]), even if A-bypass is utilized.

The real address modes are illustrated below. In summary, if A-bypass is not used, the upper 12 bits of the memory address (the page address) come from the RH/R pair named by the rB field, while the lower 8 bits (the page displacement) are defined by the desired ALU operation. This feature can be used to combine the real-page number, as read from the Map in the previous cycle, with a displacement into the page.

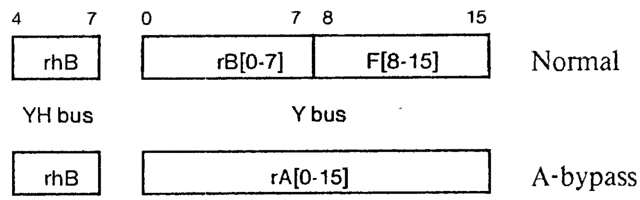


Figure 8. MAR Address Types

MAR← Effect: pageCross Branch

The second effect of a MAR← is that it automatically specifies a pageCross branch: 1 is *or'd* into INIA[10] if the ALU operation results in a carry out from the low half. Thus, although the carry out from the low byte does not propagate into the high byte, as discussed above, it can be detected as a transfer of control. A true pageCross branch can imply that the real address is invalid and that a remapping of the virtual address which originally generated it is necessary. Since pageCross is not *or'd* into INIA[11], other simple branches can be concurrently specified.

pageCross is defined to be (pageCarry *xor* aF[2]), where pageCarry is the carry out from the low 2901 byte. The *xor* has the effect of toggling pageCarry when doing subtraction while pageCross equals pageCarry when doing addition. The aF = (R-S) form of subtraction does not cause pageCarry to be inverted since aF[2] = 0; however, the aF = (R-S) form covers the most common subtraction requirements. See the *DMR*.

A complication of the MAR← automatic pageCross branch is that pageCross can indeed equal 1 if the 2901 executes a logical, instead of an arithmetic, function. See the *DMR*.

MAR← Effect: Cancelation of c2 Functions

The third effect is that if pageCross = 1 during a MAR←, then a following MDR←, IBDisp, or AlwaysIBDisp in c2 is ignored. This mechanism can be used to prevent writing into the wrong page or dispatching on the next Emulator instruction when the corresponding virtual address should be remapped. This effect increases the need to avoid logic functions during a MAR←. See the *DMR*.

1.4.2 Virtual Address References

When either the fX or fY fields equal Map← in cycle 1, a memory reference to the virtual-to-real, page-translation Map is caused. The Map is a table whose first entry is at location 10000 hex, just after the display bank. During a Map reference, the memory system uses the upper 16 bits of the virtual address (14 bits in the case of a 22-bit virtual address) to index into the table. Each entry of the table contains a 12-bit real-page number and four flags pertaining to the virtual page. Currently, a 16K table is used by the Emulator. Figure 10 illustrates the process.

The virtual address is made available to the Memory Control card on the YH and Y buses. The low byte of the Y bus is ignored and, unlike MAR←, there are no ALU side effects. Since the Y bus can be driven from either the 2901's F bus or A-bypass, addresses can be either pre or postmodified:

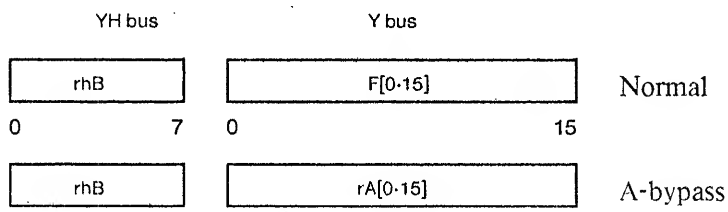


Figure 9. Map Address Types

For 24-bit virtual references, all of the YH bus is used. However, with early versions of the CP, which assumed a maximum 22-bit virtual address, if either YH[0] or YH[1] are 1, an Error trap resulted.

The following figure shows the format of a Map entry. See the *DMR* for a description of how the referenced, dirty, and present Map flag bits are maintained.

The mem field should not be set in c1 along with a Map← unless MAR←'s side effects are explicitly desired. Moreover, if YH[4-7] = 0, such clicks will be suspended due to display bank contention.

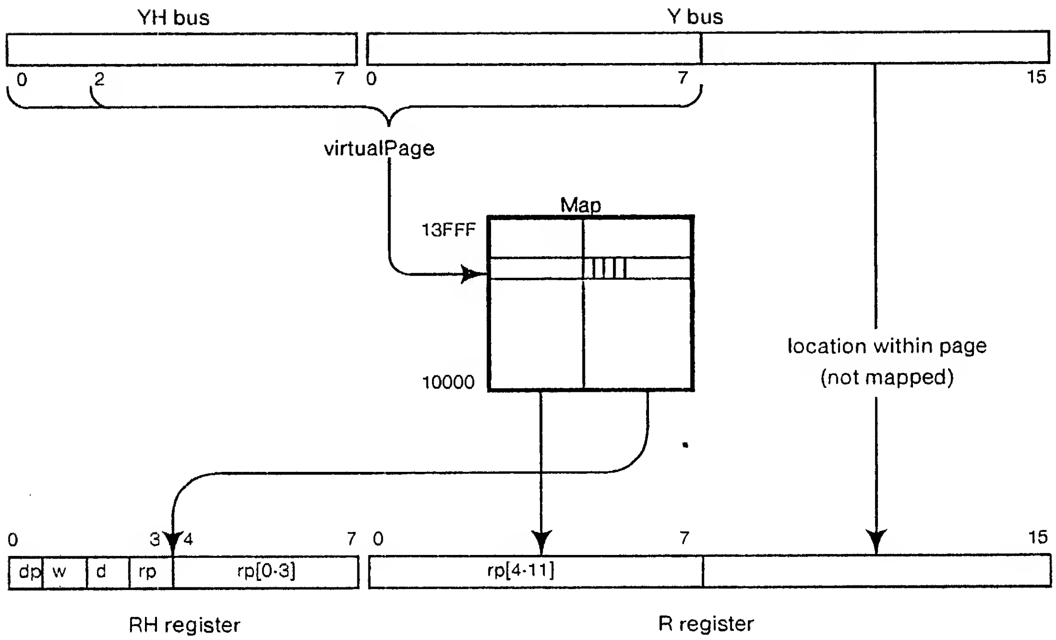
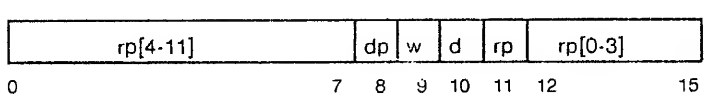


Figure 10. Virtual to Real Address Mapping



rp[0-11]	Real Page Number
dp	Dirty & Present flag
w	Write Protect flag
d	Dirty flag
rp	Referenced & Present flag

Figure 11. Map Entry Format

1.5 CP Control Architecture

This chapter discusses the algorithms used for controlling the execution of microinstructions and the interface between the IOP and the CP. Figure 12 is a block diagram of the control paths and registers.

As presented in the introduction, cycles are illimitably executed c1, c2, and c3. Every cycle, one microinstruction is decoded and executed while the next is being read from the control store (except in those clicks which have been suspended due to display bank contention). Since a device task does not execute in consecutive clicks, there is hardware to save the microprogram counter of each task while it is not running.

We first look at branching, dispatching, the Link registers, and the Error traps, as they can be described without reference to the tasking structure.

1.5.1 Conditional Branching and Dispatching

Every microinstruction can potentially branch: during each cycle, condition bits specified by the executing microinstruction are *or'd* into the next instruction's "goto"-address field (INIA) being read from control store. At the end of the cycle, this results in an address (NIA) which is used to read the next microinstruction. If the executing microinstruction does not specify a branch function, then 0 is *or'd* into INIA and, accordingly, a branch does not occur. When a microinstruction specifies a dispatch function, up-to-four bits are *or'd* into the INIA field; selecting one of up-to-sixteen target microinstructions. (The maximum of four dispatch bits was chosen in order to minimize the number which must be saved between task switches.)

Thus, all branches and dispatches take two cycles to complete: one cycle to specify the branch and one to read out the target microinstruction. The microinstruction bits required to specify a branch are fS[0-1] = DispBr and the fY field which names the branch or dispatch (Figure 13).

The notation used to specify the branching behavior is as follows: A microinstruction is located in control store at its Instruction Address, IA; the Next Instruction Address, NIA, is the control store address register; and the Intermediate Next Instruction Address, INIA, is the 12-bit "goto" address present in each microinstruction. Every cycle, the hardware *or's* the condition bits specified by fY (abbreviated DispBr) and together with a Link register specified by fX into INIA, thereby producing the NIA value used for the next cycle:

NIA[0-11] ← INIA[0-11] or DispBr[0-3] or Link[0-3].

In the case of dispatches, it is not always necessary for the microcoder to provide target instructions for each possible outcome. Any particular condition bit can be ignored by placing a 1 in its corresponding position in INIA. This method can also be used to cancel unwanted, pending branches. See the *DMR*.

Figure 13 enumerates the available branches and dispatches. Note that, in some cases, there is more than one way to branch on a particular bit and that any bit on the low half of the X bus can be branched on. The NZeroBr exists so that code can be more readily shared.

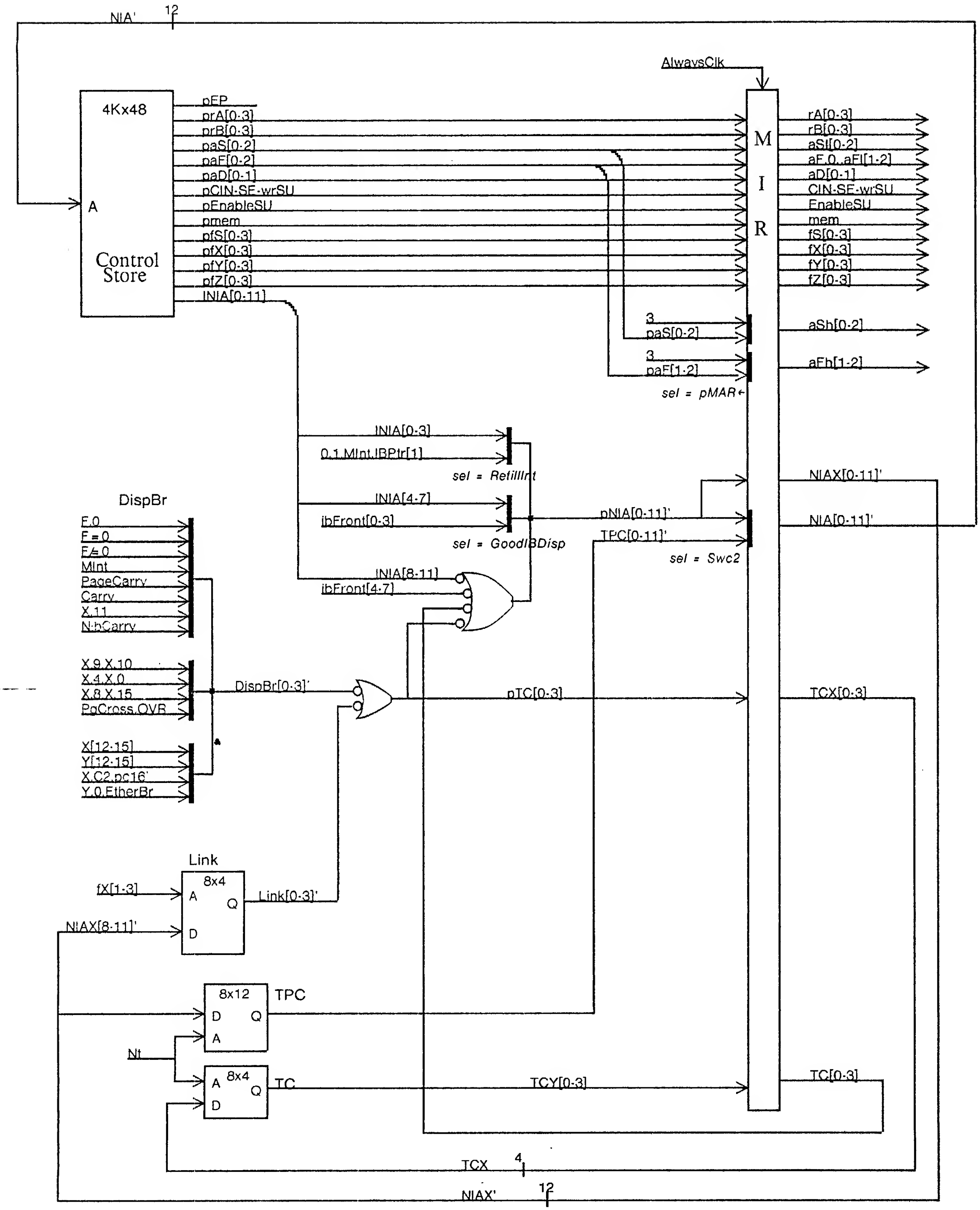


Figure 12. CP Control Paths

	source	INIA	
NegBr	F[0]	11	sign of alu result (not necessarily Y[0])
ZeroBr	F=0	11	alu output equal to zero
NZeroBr	F≠0	11	alu output not equal to zero
CarryBr	Cout[0]	11	alu carry out
NibCarryBr	Cout[12]	11	alu carry out from low nibble
PgCarryBr	Cout[8]	11	alu carry out from low byte
XRefBr	X[11]	11	present & referenced Map bit
MesaIntBr	MInt	11	Emulator Interrupt (see 1.5.3)
XwdDisp	X[9],,X[10]	[10-11]	write protect & dirty Map bits
XHDisp	X[4],,X[0]	[10-11]	X (high) bus
XLDisp	X[8].,X[15]	[10-11]	X (low) bus
PgCrOvDisp	PgCross,,OVR	[10-11]	pageCross & alu overflow
XDisp	X[12-15]	[8-11]	low nibble of X bus
YDisp	Y[12-15]	[8-11]	low nibble of Y bus
XC2npcDisp	X[12-13],,c2,,~pc16	[8-11]	X bus, cycle2, inverse of pc16
YIODisp	Y[12-13],,bp[39],,bp[139]	[8-11]	I/O branches (bp = backplane pin)
IBDisp	ibFront	[4-11]	Instruction Buffer
LnDisp	Linkn	[8-11]	Link register (n = 0..7)

Equivalent names: EtherDisp = YIODisp, XDirtyDisp = XLDisp.

Figure 13. Branches and Dispatches

1.5.2 Instruction Buffer Dispatch

The instruction buffer dispatch, IBDisp, is a special dispatch since more than four bits are *or'd* into INIA. Consequently, IBDisp can only occur in c1 or c2, and, by convention, it is restricted to c2. See section 1.3.5 for a discussion of the instruction buffer.

Assuming that the instruction buffer is full, IBDisp can cause a 256-way dispatch based on the value of ibFront: NIA[4-7] is set to the high nibble of ibFront and the low nibble of ibFront is *or'd* with INIA[8-11]. (Due to the *or* operation into the low nibble of INIA, simultaneous Link register dispatches are possible.[6]) INIA[0-3] is unaffected by the IBDisp (except by the four IB-Refill trap values); therefore, up-to-twelve 256-way dispatch tables can be concurrently used.

If the buffer is not full (ibPtr ≠ full) when an IBDisp is executed, or there is a pending interrupt, then an IB-Refill trap occurs (See 1.5.5.1).

A special version of IBDisp, called AlwaysIBDisp, never IB-Refill traps: AlwaysIBDisp dispatchs on ibFront even if there is a pending interrupt (MInt = 1) or the buffer is not full. It is used in the Emulator refill and jump microcode (see 1.6.4) to dispatch on ibFront while the buffer is still being filled. AlwaysIBDisp is encoded as fY = IBDisp and fZ = IBPtr←1.

If the microinstruction executed before an IBDisp or AlwaysIBDisp causes an IB-Empty Error trap, or it contains a MAR← and the 2901 computation results in pageCross = 1, then the IB dispatch (or possible IB-Refill trap) does not occur and ibPtr remains unaffected. Since INIA is not modified in this case, control transfers to the first entry of the macroinstruction dispatch table. (Accordingly, Emulator opcode 0 should not be assigned to a macroinstruction.)

1.5.3 MInt Register

The 1-bit MInt register can be used to interrupt the contiguous execution of Emulator macroinstructions. When MInt is set in a antecedent cycle, IBDisp traps instead of dispatches (1.5.5.1). MInt is set with fY = MesaIntRq and cleared with fY = ClrIntErr. (ClrIntErr also resets the EKErr register.) See the *DMR* for user conventions.

1.5.4 Link Registers

The CP has eight, 4-bit Link registers which can be loaded from the low four bits of the control store address. Generally, these Link registers can be used to hold four bits of state information derived directly from the flow of control. Thus, previously determined state information can be easily recalled by dispatching on a Link register. Moreover, macroinstructions can share common code at various stages of their execution and Link registers can be used for subroutine call and return structures. See the *DMR*.

The Link register addressed by fX is written with the low nibble of NIAX (which equals NIA except during a task switch in c2. see 1.5.6.4). A Link register is written when fX is in [0..7] and NIA[7] = 0: Link[fX] ← NIAX[8-11].

A Link register is *or'd* into the low nibble of INIA when fX is in [0..7] and NIA[7] = 1, causing a potential 16-way dispatch. Since the Link register is designated by an fX function, the fY field is free to specify other condition bits which can be *or'd* into INIA[8-11].

If the preceding microinstruction does not specify a branch or dispatch condition, then the Link register is loaded with a constant. However, if the prior instruction contains a branch or dispatch, the value loaded depends on the outcome of the branch or dispatch. (The low four bits of the IB dispatch value can also be recorded in this way.) See the *DMR*.

1.5.5 Microcode Traps

There are two general classes of microcode traps: IB-Refill and Error. The former only occurs as the result of IBDisp's; hence between the execution of macroinstructions. There are four IB-Refill trap locations which are a function of ibPtr and MInt. Error traps can occur in any cycle and always trap to location 0 in c1. The Error traps have priority over the IB-Refill traps and cannot be disabled.

1.5.5.1 IB-Refill Traps

If an IBDisp is executed and ibPtr ≠ full or MInt = 1, then the ibFront dispatch does not occur and instead an IB-Refill trap is caused. Specifically, ibPtr is unaffected, INIA[4-11] is not modified, and NIA[0-3] is set to the 4-bit quantity 0,,1,,MInt,,ibPtr[1]. The following table summarizes the interpretation of the IB-Refill trap locations. (If an IB-Refill trap occurs and MInt = 0, then ibPtr can not equal full.)

NIA[0-3]	MInt	ibPtr
4	0	empty
5	0	not empty (i.e., byte or word)
6	1	empty or full
7	1	byte or word

AlwaysIBDisp never IB-Refill traps and a MAR← caused pageCross branch or IB-Empty Error trap cancels a potential IB-Refill trap.

1.5.5.2 Error Traps

Error traps can result when one or more predefined error conditions are detected in the CP or memory. All error traps cause the instruction at microstore location 0 to be executed in c1 by the Emulator or Kernel, depending on the error type. Error traps cannot be disabled.

The EKErr register, read onto X[8-9] with ←ErrnIBnStkp, names the type of error:

EKErr	Type
0	control store parity error
1	Emulator memory error
2	stackPointer overflow or underflow
3	IB-Empty error

If, coincidentally, two or more errors occur at the same time, smaller values of EKErr are reported. The error types are also accumulated until EKErr is reset: the minimum value is reported when EKErr is read. Error traps have priority over the IB-Refill trap. See the *DMR* for example error-handling microcode.

EKErr is reset by the ClrIntErr function which, as a side effect, also resets any pending interrupts.

With early CP modules, an EKErr value of 1 can also imply that a 23- or 24-bit virtual address had been used by the Emulator. In this case, the ErrorLogging register in the Memory Controller is read to determine whether the error is actually a double-bit memory error. Since the Memory Controller can now accept 24-bit virtual addresses, this interpretation of EKErr = 1 is no longer necessary (beginning with CP etch 4, Rev N).

CS Parity Error

If the parity of a microinstruction read by any task is odd, then control is transferred to location 0 at the Kernel task level. Since the Kernel is the highest priority task, no other microcode tasks can execute. The CS-parity-error signal is sampled by the IOP, which can consequently sense a failed control store chip.

If the instruction read from microstore in c1 has bad parity, then the Kernel runs at location 0 in the next c1. If the parity error occurs in c2 or c3, then there is a one click delay before the Kernel executes at location 0 in c1. This intervening click can be executed by any task.

Emulator Memory Error

If the Memory Controller indicates a double-bit memory error in c3 during an ←MD executed by the Emulator, then a trap to location 0 in c1 occurs at the Emulator task level.

The hardware requires the execution of one additional Emulator click between the c3 which errored and the trap at location 0. Thus, other tasks and an additional Emulator click can intervene between the occurrence of the error and the trap code.

This trap only occurs for memory errors incurred by the Emulator task: device tasks must explicitly utilize the ErrorLogging register in the Memory Controller. Yes, the memory address is lost (as well as the syndrome if other memory reads occurred since the error).

Stack Pointer Overflow or Underflow

If a pop or push is executed with the values of the stackPointer given in the following table, then a trap to location 0 in c1 at the Emulator task level occurs (the stackP is still modified).

The hardware requires the execution of one additional Emulator click before the trap at location 0. Thus, other tasks and an Emulator click can intervene between the occurrence of the error and the trap code.

Multiple pop's and push's can be specified per microinstruction in order to ameliorate the detection of Stack overflow or underflow. For instance, fXpop (i.e., the pop in the fX field), fZpop, and push executed together leave the stackPointer unmodified, yet simulate two pop's with respect to stack underflow detection. fXpop with push checks for stack overflow while not moving the stackPointer, and, likewise, push and fZpop check for underflow. The following table enumerates the cases.

functions	stackP	Trap is	if stackP is
pop	-1	underflow	0
push	+1	overflow	15
fXpop, push	0	underflow	0
push, fZpop	0	overflow	15
fXpop, fZpop	-1	underflow	0 or 1
fXpop, fZpop, push	0	underflow	0 or 1

If the Emulator top-of-stack (TOS) element is kept in an R register and the rest of the Stack in the U registers, and it is assumed that TOS can always be stored away into the Stack, then these values imply a maximum stack size of 14 words.

IB-Empty Error

If an ←ib, ←ibNA, ←ibLow, or ←ibHigh is executed when ibPtr = empty, then an IB-Empty Error trap occurs to location 0 in c1 at the Emulator task level. If the IB-Empty Error occurs in c1, a MDR← in the next cycle is canceled. (Furthermore, an IBDisp is ignored, but this fact is of no particular value.)

In normal operation (sec. 1.3.5) the IB is always guaranteed to have enough operand bytes (two) before a macroinstruction begins executing. However, when the macroprogram counter points to the last word of a page, the buffer is intentionally not refilled by the Emulator "refill" microcode and the IB-Empty trap can occur, indicating that control has actually proceeded across a page boundary. See the *DMR*.

If the IB-Empty error occurs in c1, then control transfers to location 0 in the next Emulator c1. However, if the error occurs in c2 or c3, the hardware requires the execution of one additional Emulator click before the trap at location 0. Consequently, other tasks and an Emulator click can intervene between the occurrence of the IB-Empty error in c2 or c3 and the trap code. In particular, if such a click executed a MDR← with an address which was a function of an IB value read in the previous c2 or c3, then a random memory location can be written.

The IB is not read during c2 or c3 of a macroinstruction's last click. However, the microcoder must ensure that, immediately following an ←ib, ←ibNA, ←ibLow, or ←ibHigh function executed in c2 or c3, there is not a memory write with a MAR← or Map← address which is a function of the IB value read in c2 or c3. (This is not checked for by MASS.)

1.5.6 Task Scheduling and Switching

A task is the microcode which supports an IO device or the Emulator. A device task runs whenever the device controller in the Dandelion asserts its "wakeup" request. Since a device task can only run during its pre-allocated clicks, a controller's maximum memory latency and maximum memory bandwidth is an outcome of its preassigned location within the round.

The Emulator and Kernel tasks behave differently than device tasks. The Kernel task is a special task used for communication between the CP and IOP (see 1.5.6.6). The Emulator task has no fixed assigned slot in the round: it executes during a click which a controller has opted not to use. Since devices do not utilize all of the bandwidth implied by the round structure, there is always a minimum number of clicks available to the Emulator.

1.5.6.1 Task Allocation

The CP can control a maximum of 8 tasks. Currently, there are 6 wakeup lines (5 of them on the backplane) which can request microcode service. The eight task numbers are allocated between the devices, Emulator, and Kernel as follows:

0	Emulator
1	Display or LSEP or MagTape
2	Ethernet
3	Refresh (Auxiliary)
4	Disk (Rigid)
5	IOP
6	IOP control store read/write address
7	Kernel

The Dandelion is configured at boot time so that either the Display, or the LSEP, or the MagTape can use task number 1, but all three can not simultaneously use task 1. Normally, the Display task controls the refreshing of memory, but when the LSEP or MagTape (or other Option board controller) is active instead of the Display, then the Refresh task has this responsibility. Similarly, the Disk task cannot be simultaneously used by both the SA4000 and SA1000. Task 6 is currently not assigned to an actual device: instead it is used by the IOP as an address register when reading or writing the control store (see 1.5.6.7).

1.5.6.2 Click Allocation

There are two types of rounds: a standard 5-click round and an extended 10-click round. The standard round is used with the HSIO-I board (Shugart SA4002 or SA1002 disks) and the extended round with the HSIO-II board (LDC, or LargeDiskController: Trident or Hunter drives). The extended 10-click round is an "even" 5-click round followed by an "odd" 5-click round. In the even rounds, the Ethernet task has claim to click 3, and in the odd rounds the Trident disk controller does.

Click 4 is special because the Display Controller hardware guarantees that memory references to the display bank can never abort in this click. In order to refresh memory and maintain the cursor, the Display and Refresh tasks are assigned to this click. The LSEP also uses this click as its band buffers are located in the Display Bank. Within click 4, the LSEP or MagTape controllers have priority over the Refresh task.

The following tables show the standard and extended rounds:

Standard Round:

Click	Task
0	Ethernet
1	SAx000 Disk
2	IOP
3	Ethernet
4	Display OR (Refresh OR (LSEP OR MagTape))

Extended Round:

Click	Task
0-0	Ethernet
0-1	Trident-Hunter Disk
0-2	IOP
0-3	Ethernet
0-4	Display OR (Refresh OR (LSEP OR MagTape))
1-0	Ethernet
1-1	Trident-Hunter Disk
1-2	IOP
1-3	Trident-Hunter Disk
1-4	Display OR (Refresh OR (LSEP OR MagTape))

1.5.6.3 Click Bandwidth Utilization

The following table summarizes the bandwidth availble to each device and the percentage which remains for the Emulator when the controller is transferring data. (Pre- and post-data-transfer overhead, which normally utilizes 100% of device clicks, is not included.) Note that the IOP only transfers one byte per click, so its maximum available rate is actually 3.9 mbits/s.

Device	BW allocated (mbits/s)	BW used (mbits/s)	% remaining for Emulator
Ethernet(w/SAx000)	15.6	10.0	36
Ethernet(w/Trident)	11.7	10.0	15
SA4000	7.8	7.14	8.5
SA1000	7.8	4.27	45
LDC (Trident/Hunter)	11.7	9.6	18
Display (microcode)	7.8	1.1	86
IOP	7.8	2.0	26
LSEP-Refresh	7.8	3.7 + 1.1	38
MagTape-Refresh	7.8	.6 + 1.1	78

Even with the Ethernet, SA1000, and IOP concurrently transferring data and the Display microcode refreshing memory, the Emulator still executes 60% of the time.

1.5.6.4 Tasking Hardware

The CP control hardware was designed to hide the details of task switching from the programmer. Since tasks are frequently resumed and suspended by controller wakeup requests, the hardware performs all the necessary start upand stop functions: every click it saves the current task's microprogram counters and pending condition bits and, when it is scheduled to run again, it restores them. Figure 14 illustrates the process, outlined below.

Every c2 the Schedule Prom in the CP, on the basis of the controller wakeups and click number, decides which task (Nt) will run in the next click. Also in c2, the Switch Prom, on the basis of Nt, the currently executing task (Ct), and Wait (x.xx), decides whether to activate the task switching logic (and, if so, sets Swc2 ← 1). A task switch has two parts dealing with the outgoing and incoming microprogram counter and conditions: (1) a restore process and (2) a save process.

(1) The Temporary Program Counter (TPC) array holds the eight 12-bit task microprogram counters. If it is cycle 2 and a task switch is occuring, the TPC, as addressed by the next task number, is the source of the control store address. The next task's first micronstruction is subsequently read in c3 and executed in the following c1. In short, NIA ← TPC[Nt] at the end of c2.

At the same time the next task's microprogram counter is being read from TPC[Nt], the saved condition bits are read out of the Temporary Conditions array, TC, and latched into the TC regsiter. During c3, TC is *or'd* with the next task's first microinstruction INIA field, which is being read from the microstore. In summary, the saved condition bits are read during c2 from TC[Nt], latched into the TC register, and in c3 *or'd* with INIA.

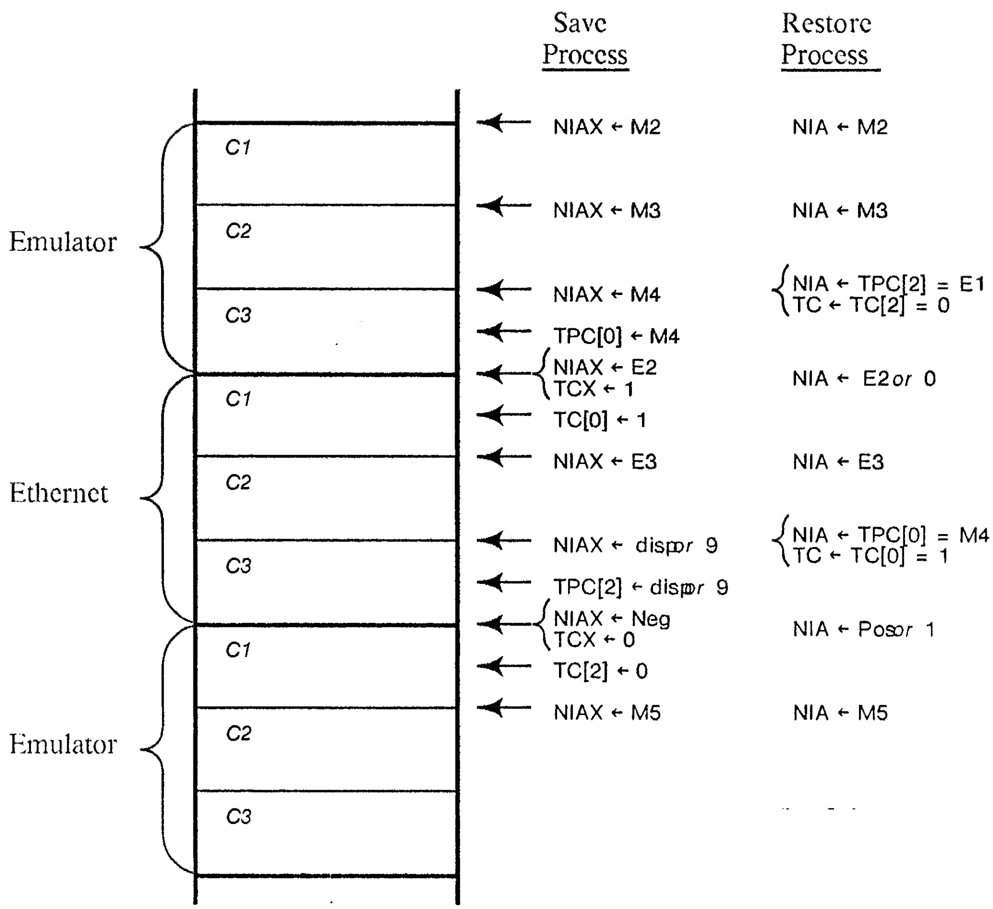
(2) The current task's Next Instruction Address (which would have been loaded into NIA if there were no task switch) is latched into the NIAX register at the end of c2 and then saved in the current task's TPC location during c3. In general, every c3, TPC[Nt] ← NIAX. (Note that in c3, Nt equals the task currently executing.)

Furthermore, the condition bits of the task currently executing (which would have been *or'd* into INIA) are latched into the TCX register at the end of c3 and then saved into the TC array in c1. In general, every c1, TC[Nt] ← TCX. (In c1, Nt actually equals the task which executed in the previous click. The condition bits are saved in c1 because there is not enough time in c3 to write them into a RAM.)

The following table summarizes when the TPC and TC are read and written and the interpretation of Nt:

cycle	operation	Nt
end of c2	NIA ← TPC[Nt]	next task
c3	TPC[Nt] ← NIAX	current task
end of c3	NIA ← INIA *or* TC	
end of c3	TCX ← DispBr *or* Link	
c1	TC[Nt] ← TCX	previous task

The TPC and TC RAMs are written every click (except suspended clicks) even if there is not a pending task switch. Consequently, if the Emulator is suspended because of Display bank interference, it's correct restart address is available in the TPC.



{Emulator microcode for above example.}

M1:	Noop,	c1;
M2:	Noop,	c2;
M3:	[] ← -1, NegBr,	c3;
M4:	BRANCH[Pos, Neg],	c1;
Pos:	GOTO[ME]	c2;
Neg:	Noop	c2;
M5:	Noop	c3;

{Ethernet microcode for above example}

E1:	Noop	c1;
E2:	XBus ← 9, XDisp,	c2;
E3:	DISP4[disp],	c3;

Figure 14. Demonstration of Tasking Mechanism:

Where the Emulator task (0) is preempted by the Ethernet task (2) for one click.

This example demonstrates a pending branch across the task switch for the Emulator and shows when the TPC and TC arrays are written and when NIAX is not equal to NIA.

The Save Process refers to the writing of the TPC & TC arrays, while the Restore Process refers to the reading out of TPC & TC.

1.5.6.5 Display Bank Interference

If any task references the dual-ported Display bank (lowest 64K of real memory) and the Display controller is reading the bank, then the task is suspended for the duration of that click; that is, no microinstructions are executed during the suspended click. Click suspension is always in multiples of clicks and the c1-c2-c3 structure is not modified. Device tasks should not reference the Display bank (unless the Display is off).

In particular, the Emulator task is suspended until either it is scheduled for click 4 or the Display controller relinquishes the low bank. This reduces by 60% the Emulator's maximum possible bandwidth into the low bank when the Display is active: from 38.4 to 15.8 mbits/s (1 megaword/s).[7]

Clicks are suspended by the signal Wait which gates off all clocks which can change sensitive state information. In the schematics, such clocks are labeled WaitClock, in contrast with the normal AlwaysClock. Wait is defined

Wait ← (MAR← *and* YH[4-7]=0 *and* Disp-Proc'=0) *or* (IOPWait *and* c1)
or (Wait *and* c2) *or* (Wait *and* c3).

When Wait is true, the following registers and RAMs are not written: R, Q, U, RH, stackP, IB[0], IB[1], ibFront, ibPtr, Link, TC, TPC, MInt, pc16', and Errors (Memory, stackPointer, CSParity, IBEmpty). By contrast, the following are unaffected by Wait: MIR, NIA, NIAX, TCX, TC, KernelReq, EKErr, and schedular task states (Nt, Ct, Pt, Swc3).

Since the Microinstruction (MIR) and Next Instruction Address registers' (NIA) clocks are unaffected during suspended cycles, the decoded signals from the MIR can change during an aborted click. However, this does not result in a random sequence of decoded microinstructions: the MIR output in c1, c2, and c3 is equal to the values it would have had if the click were not suspended. This is because the microinstruction loaded into MIR is always defined by an NIA which is unaffected by any invalid states generated during the suspended click: cycle 1's MIR output is defined by the NIA read from the TPC (in c2), cycle 2's by the value of INIA *or* TC (computed in c3), and cycle 3's by INIA *or'd* with conditions bits specified in c1 (which are not effected by WaitClock in c1). Furthermore, if the Emulator is suspended for consecutive clicks, the MIR output is the same for each click since NIA is reloaded from the TPC during suspended clicks.

1.5.6.6 Kernel Task

The Kernel task is used for supporting the debugging of the CP (e.g., breakpoints, reading/writing CP registers) and handling the CP-IOP communication while booting (e.g., memory refresh during control store read/write). When the Kernel task is enabled, it executes in all clicks, preempting all device tasks and the Emulator.

The Kernel task runs if there is a CSParityError, IOPWait is true (1.5.6.7), or the microcode function EnterKernel is executed. If EnterKernel is executed in c1, the Kernel runs in the next click. However, if executed in c2 or c3, an Emulator or device click can intervene before the Kernel runs. When the Kernel task is started, the Switch Prom does not cause a task switch; hence, a breakpoint microinstruction can specify an entry point into the Kernel.

The Kernel task request remains active until reset by the ExitKernel function. An ExitKernel is overridden by a pending IOPWait or CSParityError. When ExitKernel is executed in c1, the next click can be executed by another task (depending on which click the ExitKernel is in and the wakeup requests).

1.5.6.7 CP-IOP Interface

The IOP interfaces with the CP as both a standard I/O controller and as a boot loader/debugger. This section deals with the booting interface: the control lines used to load the control store and initialize the tasks' microprogram counters (TPCs).

The following signals are used between the IOP and CP:

SwTAddr	high level causes Nt = IOPTPCHigh[0-2] and NIAX[0-4] = IOPTPCHigh[3-7] and NIAX[5-11] = IOPData bus
IOPWait	high level sets Kernel wakeup request and WaitClock is suspended
WrTPCHigh	positive edge writes IOPTPCHigh with IOPData bus
WrTPCLow	pulse causes TPC[Nt] ← NIAX
CSWE[n]'	pulse writes a control store byte with IOPData bus
ReadCSEn'	places CS byte, TPC, & TC onto IOPData bus
ReadCS[n]	selects CS, TPC, & TC bits to use with ReadCSEn'

The basic algorithm for reading or writing control store is to first write TPC[6] with the address of the location to be accessed and then read or write data bytes (addressed by CSWE[n]' or ReadCS[n]) while allowing the Kernel to Refresh memory if necessary. Although all of the tasks' TPCs can be initialized, the TC registers cannot be loaded by the IOP.

In general, when reading or writing a TPC location or CS byte, both SwTAddr and IOPWait must be high and the value of Nt (loaded into IOPTPCHigh) must be 6 or 7.

When SwTAddr is true, Nt and NIAX are defined by the IOPTPCHigh register instead of their normal sources. This allows the IOP to address and supply data directly to the TPC RAM.

Setting IOPWait causes the Wait line to be high. Thus, registers clocked by WaitClock cannot be loaded with spurious data while a TPC or CS location is being written. (Moreover, the CSParityError trap cannot occur.) IOPWait also sets the Kernel wakeup request so that the Kernel task runs when IOPWait is removed.

While IOPWait = 1 and Nt = 6 or 7, the Switch Prom causes a continuous task switch; that is, Swc2 is always true and NIA is set to the value of TPC[6] or TPC[7]. In this state, the Kernel microcode does not run and its TPC does not change. However, after writing one byte of control store or one TPC location, it may be necessary to refresh main memory. In this case, IOPWait and SwTaddr are reset and, since the IOPWait caused the Kernel wakeup request to be set, the Kernel begins running at the saved TPC location and executes the required number of Refresh functions or performs a function enumerated by the IOP via the normal I/O interface (e.g., ←IOPIData, ←IOPStatus).

The following table shows which control store bytes are read or written with ReadCSEn' and CSWE[n]'. Note that when writing the control store the inverse of the data must be supplied on IOPData.

ReadCS	CSWE[n]	IOPData[0-7]
0	a	rA, rB
1	b	aS, aF, aD
2	c	ep, Cin, EnableSU, mem, fS
3	d	fY, INIA[0-3]
4	e	fX, INIA[4-7]
5	f	fZ, INIA[8-11]
6		TC, TPC[0-3]
7		TPC[4-11]

1.6 Input/Output Interface

The CP and the high speed devices were mutually designed within one framework and are inexorably bound together: the I/O bus is the same as the CP's main data bus (the X bus), the I/O register control is directly encoded into the microinstruction format, and the devices depend on the preallocated click structure for guaranteed memory latency and bandwidth. This intimate relationship between the devices and the processor exists in order to absolutely minimize the overall system cost. By sharing the ALU among several controllers, overlapping memory accesses with ALU computation, and guaranteeing memory latency, very small IO controllers can be built. This section exists because it is possible to design different disk or display controllers on the HSIO board, new high speed controllers on the Option board, and new Memory systems.

1.6.1 CP-IO Interface

The following signals and buses are used between the CP and a typical device controller, called Dev:

X bus	16-bit data to or from memory or 2901
Y bus	16-bit data from 2901
DevReq'	task wakeup request to CP Schedule Prom
DevCtl←'	signal from CP to load controller control register from X or Y Bus
DevOData←'	signal from CP to load controller data register from X Bus
←DevStatus'	signal from CP to place controller status onto X Bus
←DevIData'	signal from CP to place controller data onto X Bus
ClrDevRq'	signal from CP to reset controller wakeup request
DevStrobe'	signal from CP for general use by controller
IODisp	CP branch on a controller state
Wait	level from CP to gate off WaitClock

Normal CP-Controller interaction (for input) goes something like: (1) A controller receives a word of data, (2) the controller activates its wakeup request, (3) the controller's task runs in its allocated click, (4) the microcode reads the data from the controller to main memory or 2901, and (5) the controller resets its wakeup request. In general, the wakeup request is either explicitly turned off by the task via ClrDevRq' or is turned off by the controller when it senses a ←DevIData', DevOData←', or DevStrobe'. It is explicitly assumed that a controller only causes wakeups when data transfers are pending (or when directed by its task) in order to minimize the impact on the Emulator.

A device's wakeup request must be turned off by the end of the cycle 1 which follows the service click in order to prevent a task from accidentally running again. Since the device's wakeup request must be 2-level synchronized, this implies that the reset-wakeup function must be executed in c1 or c2 for those devices which have a two-click minimum separation.

In general, all controller control registers should be clock'd with WaitClock so that spurious device actions are prevented while writing control store. If a control signal can be used by an Emulator click which could be suspended, it should also be gate'd with WaitClock. Device tasks should not reference the Display bank unless the Display is off.

1.6.2 Controller Latencies

A controller's data buffer size depends on how often the buffer is serviced and what kind of wakeup scheme is employed. There are two basic wakeup strategies: post and prerequesting. In the former case, the wakeup request is raised after the device buffer is available to be read/written by the CP. In prerequesting, the wakeup request is raised before the device buffer is actually available. Only the SAx000 disk uses prerequesting. Where a task must process some of the data and cannot transfer a word per click, then a FIFO is usually used as a buffer (as in the Ethernet). However, when little or none of the data must be examined by the microcode, then a simple register buffer is sufficient (as in the rigid disk controllers and LSEP).

In order to avoid overruns with the postrequesting scheme, the maximum microcode service latency plus the wakeup-request synchronizer delay must be less then the data rate:

$$L_{max} + s_{max} < b/r,$$

where b is the number of bits of buffering, r is the data rate of the device (in mbits/s), L_{max} is the maximum latency (in μseconds), and s_{max} is the synchronizer delay (equal to 2T, where T = .137 μsec). If the task microcode transfers one word per click, then

$$L_{max} = 3dT + 4T \text{ for output, and}$$
$$L_{max} = 3dT + 3T \text{ for input,}$$

where d is the maximum seperation between device clicks. If the microcode does not always transfer a word per click, then L_{max} is correspondingly greater.

For prerequesting, the wakeup request cannot be made too early, thus the constraint

$$s_{min} + L_{min} - t_{handoff} > 0,$$

where $t_{handoff}$ is the time for the CP to read the buffer (equal to T) or the controller to read the buffer (about .05 μsec)). If prerequesting begins p device bit times before the buffer is ready, then

$$s_{min} = 2T - p/r, \text{ and}$$
$$s_{max} = T - p/r.$$

Since L_{min} = 5T for output and 4T for input, p must satisfy the following conditions in order for prerequesting to work ($t_{handoff}$ = 0 for output):

$$[rT(3d + 6) - b] < p < 6rT \quad \text{for output, and}$$

$$[rT(3d + 5) - b] < p < 4rT \quad \text{for input.}$$

1.6.3 IO Controller Design Rules

Since replacement or augmented controllers are being designed for the Dandelion, the following design rules should be followed in order to guarantee correct operation. Figure 15 illustrates the proper application of the CP interface signals.

(1) CP control signals such as DevReq', DevCtl←', ←DevIData', ClrDevRq', and DevStrobe' originate from an SN74S138 decoder and therefore most not be used in an asynchronous way, such as the clock input of a register. These CP signals must be synchronized to the CP clock or gate'd with pAlwaysClk or pWaitClk.

(2) Controller input buffers must be either an SN74S240 or SN74S374 (or equiv) and the CP control signal which enables them onto the X bus, such as ←DevIData' or ←DevStatus', must be connected directly to the output enable input without being gate'd in any way.

(3) If there is more than one output register on the board, the X bus must be buffered with an SN74S241 (or equiv) before routed to the registers. The CP control signals which load the output registers, such as DevOData←' or DevCtl←', can be modified per the constraints of a clock qualifier signal (see (5)).

(4) The device wakeup request signal, DevReq', must come from an SN74S374 (or S74, or equiv) and must be synchronized by at least 2 such FF's.

(5) The clock qualifying structure shown in figure 8 must be used: the S02 is located in the position nearest backplane pins 1-10 and the "qualifier" gates are no further away then the center of the board, their preferred location. Clock qualifier terms should be valid by 94 nanoseconds after the positive (active) edge of AlwaysClk. Clock'd registers should be no more than 10" from their qualifier gate.

pWaitClk must be used for any register which, if spruiously loaded during a control store boot, can activate a device function (e.g., disk write enable). Such registers should also be reset by IOPReset' which is *or'd* with the power supply on/off reset.

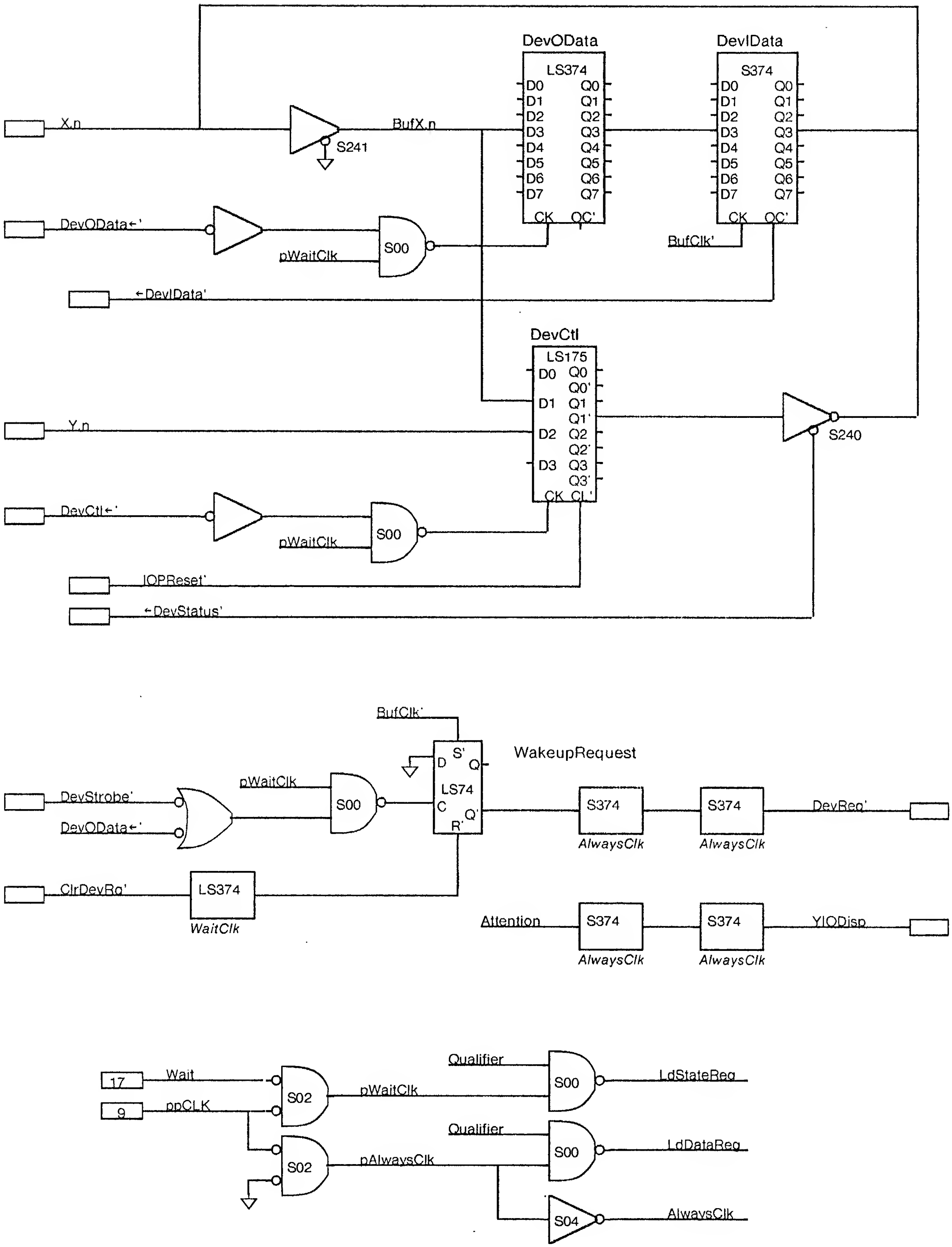


Figure 15. Controller Hardware Demonstrating I/O Rules & CP Interface

1.6.4 Example Microcode

Just as a melody, in order to be heard, requires both notes and intervals, the CP hardware should be viewed in light of its corresponding microcode. The following microcode examples illustrate how and in what time frame certain elementary functions are accomplished. There are seven examples, some simplified: Mesa Emulator Load Local n, Read n, Jump n, Refill, and the Ethernet, Disk, and LSEP inner loops. See the *DMR* for a description of the microcode format.

(1) The Mesa Emulator Load Local 1 (LL1) macroinstruction indexes the local frame pointer and then push's the addressed word from memory onto the Stack. It executes in one click if the indexing operation does not cross a page boundary and in three if a page cross occurs. If the Map flags must be updated (RMapFix), another two clicks are required.

```
@LL1:        MAR ← Q ← [rhL, L+1], L1←L1.PopDec, push,                     c1,  opcode[1'b];
LLn:         STK ← TOS, PC ← PC+PC16, IBDisp, L2←L2.LL, BRANCH[LLa,LLb,1], c2;
LLa:         TOS ← MD, push, fZpop, DISPNI[OpTable],                       c3;
LLb:         Rx ← UvL,                                                     c3;

LSMap:       Noop,                                                         c1;
             Q ← Q - Rx, L2Disp,                                           c2;
             Q ← Q and 0FF, RET[LSRtn],                                    c3;

LLMap:       Map ← Q ← [rhMDS, Rx+Q],                                      c1,  at[3,10,LSRtn];
             Noop,                                                         c2;
             Rx ← rhRx ← MD, XRefBr,                                       c3;

             MAR ← [rhRx, Q + 0], L0←L0.R, BRANCH[RMUD,$],                 c1;
             IBDisp, GOTO[LLa],                                            c2;
RMUD:        CALL[RMapFix],                                                c2;
```

(2) The Mesa Emulator Read 1 (R1) macroinstruction indexes the virtual address on the top of Stack and then push's the addressed word from memory onto the Stack. It executes in two clicks. Four are required if the page has been read the first time; that is, the Map flags must be updated.

```
@R1:         Map ← Q ← [rhMDS, TOS + 1], L1←L1.Dec, pop,                   c1,  opcode[101'b];
             push, PC ← PC + PC16,                                         c2;
             Rx ← rhRx ← MD, XRefBr,                                       c3;

             MAR ← [rhRx, Q + 0], L0←L0.R, BRANCH[RMUD,$],                 c1;
             IBDisp, GOTO[LLa],                                            c2;
```

(3) The Mesa Emulator Jump 2 (J2) macroinstruction increments the PC by 2 bytecodes and then refills the instruction buffer. It executes in two clicks. Five are required if the jump crosses a page boundary.

```
@J2:         MAR ← PC ← [rhPC, PC+1], push,                                c1,opcode[201'b];
             STK ← TOS, L2 ← L2.Pop0IncrX, Xbus←0, XC2npcDisp, DISP2[jnPNoCross], c2;

jnPNoCross:  IB ← MD, pop, DISP4[JPtr1Pop0, 2],                            c3,  at[0,4,jnPNoCross];
jnP1Cross:   Q ← 0FF + 1, L0 ← L0.JRemap, CANCELBR[UpdatePC, 0F],          c3,  at[2,4,jnPNoCross];

JPtr1Pop0:   MAR ← [rhPC, PC + 1], IBPtr←1, push, GOTO[Jgo],               c1,  at[2,10,JPtr1Pop0];
JPtr0Pop0:   MAR ← [rhPC, PC + 1], IBPtr←0, push, GOTO[Jgo],               c1,  at[3,10,JPtr1Pop0];
Jgo:         TOS ← STK, AlwaysIBDisp, L0 ← L0.NERefill.Set, DISP2[NoRCross], c2;
```

(4) The Mesa Emulator instruction buffer refill code executes in one click if the buffer was not empty and in two if it was. Four to six clicks are required if the refill occurs across a page boundary.

```
{Buffer Empty Refill. Control goes from NoRCross to RefillNE since RefillE+1 does not contain an IBDisp.}
RefillE:    MAR ← [rhPC, PC], PC ← PC-1, L0 ← L0.ERefill,               c1, at[400];
            PC ← PC+1, DISP2[NoRCross],                                 c2;

{Buffer Not Empty Refill.}
OpTable:        {"Noop" location of Instruction Dispatch table}
RefillNE:   MAR ← [rhPC, PC + 1],                                       c1, at[500];
            AlwaysIBDisp, L0 ← L0.NERefill.Set, DISP2[NoRCross],        c2;

NoRCross:   IB ← MD, uPCCross ← 0, DISPNI[OpTable],                     c3, at[0,4,NoRCross];
RCross:     Q ← 0FF + 1, GOTO[UpdatePC],                                c3, at[2,4,NoRCross];
```

(5) The Ethernet input inner loop transfers one word per click until either a page boundary is crossed (ERead+2 or ERead+3), the maximum sized packet has been exceeded (EITooLong), or the controller has signaled an abnormal condition (ERead+1 or ERead+3).

```
{main input loop}
EInLoop:    MAR ← E ← [rhE, E + 1], EtherDisp, BRANCH[$,EITooLong],     c1;
            MDR ← EIData, DISP4[ERead, 0C],                             c2;

ERead:      EE ← EE - 1, ZeroBr, GOTO[EInLoop],                         c3, at[0C,10,ERead];
            E ← uESize, GOTO[EReadEnd],                                 c3, at[0D,10,ERead];
            E ← EIData, uETemp2 ← EE, GOTO[ERCross],                    c3, at[0E,10,ERead];
            E ← EIData, uETemp2 ← EE, L6←L6.ERCrossEnd, GOTO[ERCross],  c3, at[0F,10,ERead];
```

(6) The SAx000 disk write and verify inner loop transfers one word per click until the required number of words have been sent.

```
WrtVerLp:   MAR ← [RHRCnt, RCnt], RCnt ← RCnt+1,                        c1, at[0,2,FinWrtVer];
            RAdr ← RAdr-1, ZeroBr, CANCELBR[$, 2],                      c2;
            KOData ← MD, BRANCH[WrtVerLp, FinWrtVer],                   c3;
```

(7) The LSEP output inner loop outputs a band buffer entry from the display bank and then clears the entry. This continues until the required number of words have been transferred, which is detected by aligning the data on a page boundary.

```
scan:       MAR← [displayBase1, rX+0], ClrDPRq,                         c1;
            MDR← rY{= zero}, rX← rX+1, PgCarryBr,                       c2;
            POData← MD, BRANCH[scan, endLine],                          c3;
```

1.6.4 Footnotes

[1] All of the microcode-related specifications and rules presented in this chapter are validated by the microcode assembler and control-store-allocation program (MASS).

[2] The writeable control store is expensive: out of the 160 chips total, 70 are microstore chips.

A special version of the CP has been built which has a 16K control store partitioned into four, 4K banks. The 2-bit Bank register can be loaded from NIAX with fZ = Bank←. All non-Emulator tasks are forced to execute from bank 3. Error trap location 0 exists in each bank.

[3] Where did this (prime) number come from? All system timing is based on the Display's bit time, 19.59 nS (51.04 MHz, ± .05%). There are 7 bit times in a cycle and 210 cycles (14 rounds) in one horizontal display line. More precisely, the cycle time is 137.14 ± .57 nsec.

Alternatively, the cycle time (137) equals the inverse of the fine structure constant: a fundemental dimensionless constant equal to 2π times the square of the electron charge in electrostatic units, divided by the product of the speed of light and Planck's constant ($2\pi e^2/c\hbar$) !

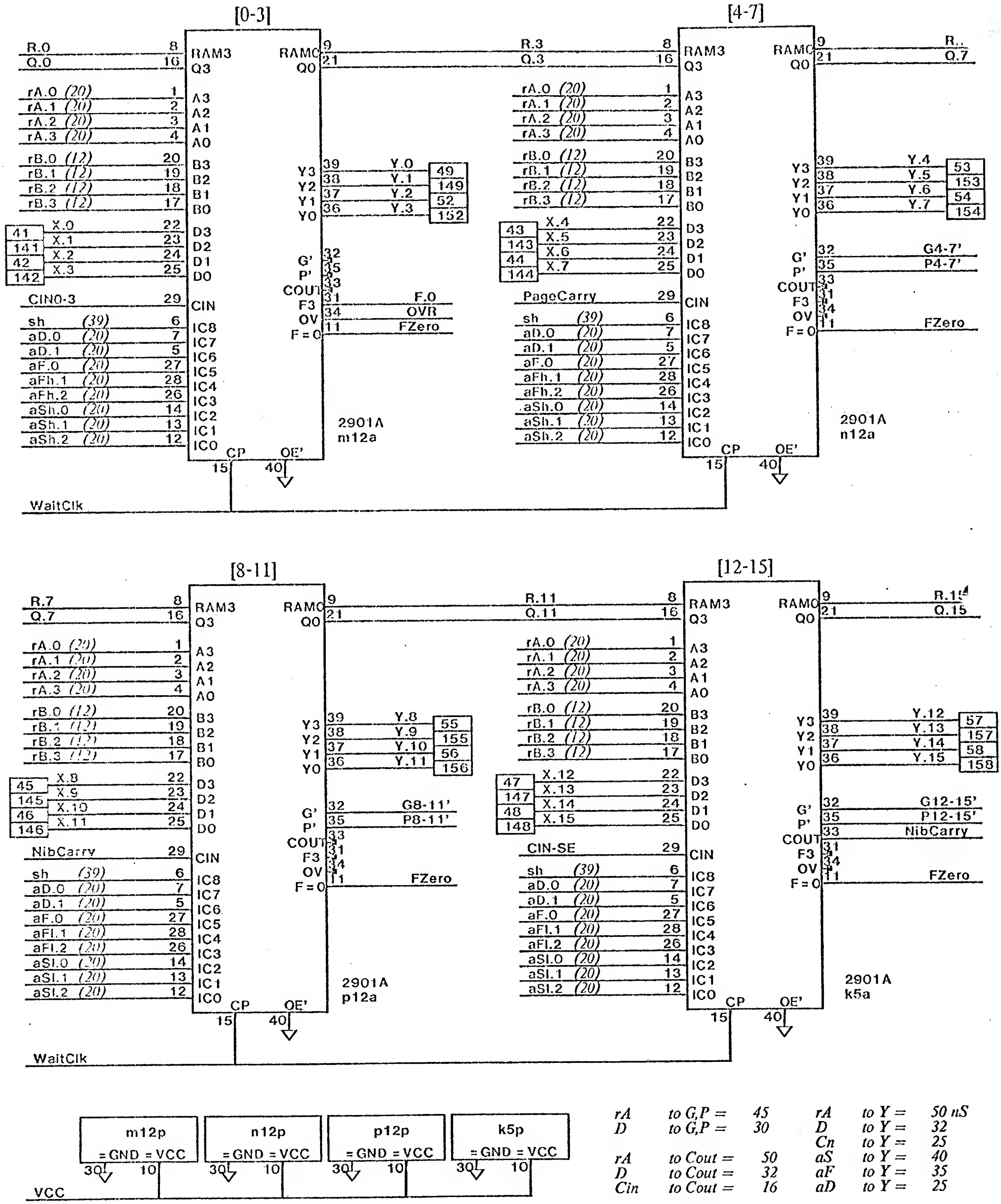
[4] This sequence has been likened to the triple time meter of a waltz!

[5] Because there are so many sources and sinks on the X bus, it has a nonnegligible capacitance: it has been measured at 337 pF!

[6] The *or*ing of a Link register with the low 4 bits of the IB byte during an IBDisp is not encouraged as this feature will not exist in a future version of the processor.

[7] The 15.8 mbits/s into the display bank is approximated as follows: There are 70 clicks per display scan line and, of these, the Display controller uses 4*11 = 44 clicks for a normal scan line. Furthermore, the display microcode uses 2 clicks for memory refresh. During 808 of the total 897 scan lines, the display controller is actually pumping bits out to the monitor. Thus, the Display controller and microcode use about (808/897)(46/70)(38.4 mbits/s) = 22.6 mbits/s of the bandwidth, leaving 38.4-22.6 = 15.8 mbits/s for the Emulator.

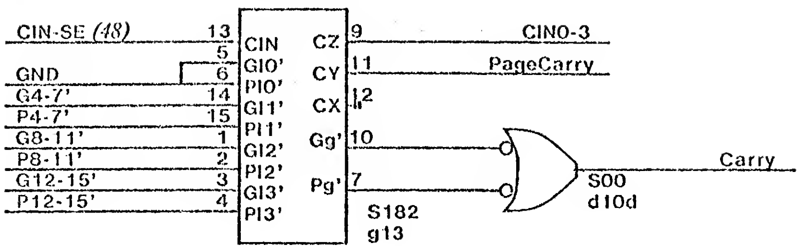
Dandelion Central Processor



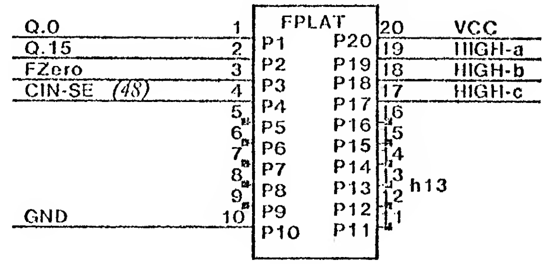
rA	to G,P =	45	rA	to Y =	50 nS
D	to G,P =	30	D	to Y =	32
			Cn	to Y =	25
rA	to Cout =	50	aS	to Y =	40
D	to Cout =	32	aF	to Y =	35
Cin	to Cout =	16	aD	to Y =	25

See Pages 29 to 34 for ALU timing

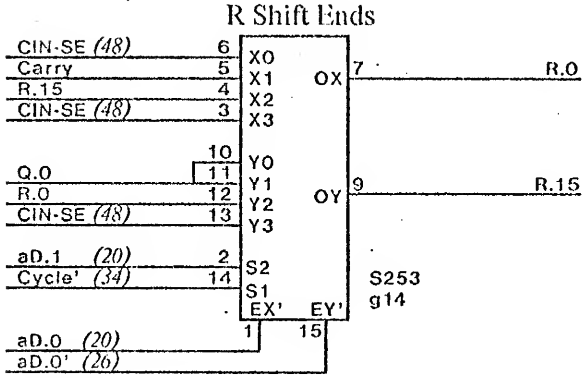
2901 Carry Lookahead



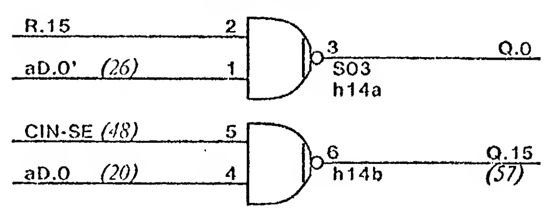
330-220 Pullups

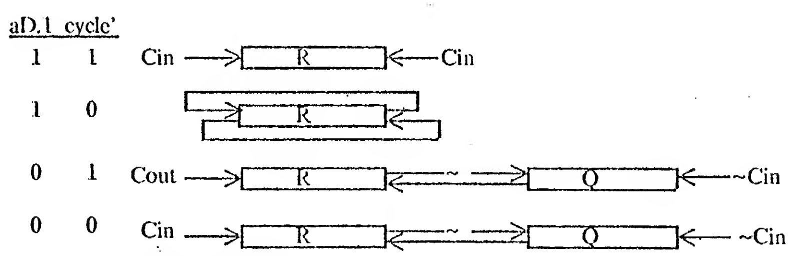


R Shift Ends



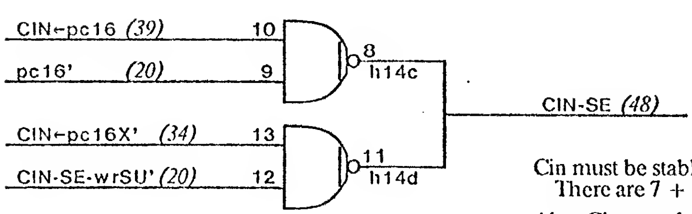
Q Shift Ends





aD.0 = 0 implies right shift

Cin & Shift Ends



Cin must be stable before 32 - 25 = 7 nS after the X-bus is stable for X-bus arith.
There are 7 + (60-48) = 19 nS to spare.

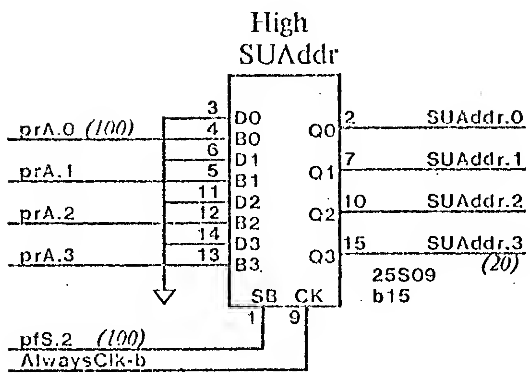
Also, Cin must be stable 50-25 = 25 nS after rA or rB are stable for register arith.
Since this is not the case for rA or rB, register arithmetic timing in bits [8-15] is dependent upon Cin timing and not rA timing.

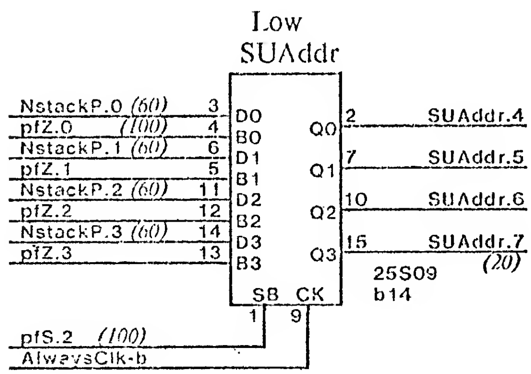
XEROX SDD	Project Dandelion	Shift Ends, Cin, YBus	File LionHead02.sil	Designer Garner	Rev C	Date 10/30/79	Page 02

SU X-bus disable

15[3]	↑ to CIN-SE-wrSU (tPLH)
30	Output Disable
10	X-bus

55[3] = 58 nS





XBus ← SU = max(75,60) nS

17[3]	↑ to SUAddr
45	tAA
10	X-bus

72[3] = 75 nS

17[3]	↑ to CIN-SE-wrSU/EnableSU
30	F93422 OE'/CE2 to X-bus
10	X-bus

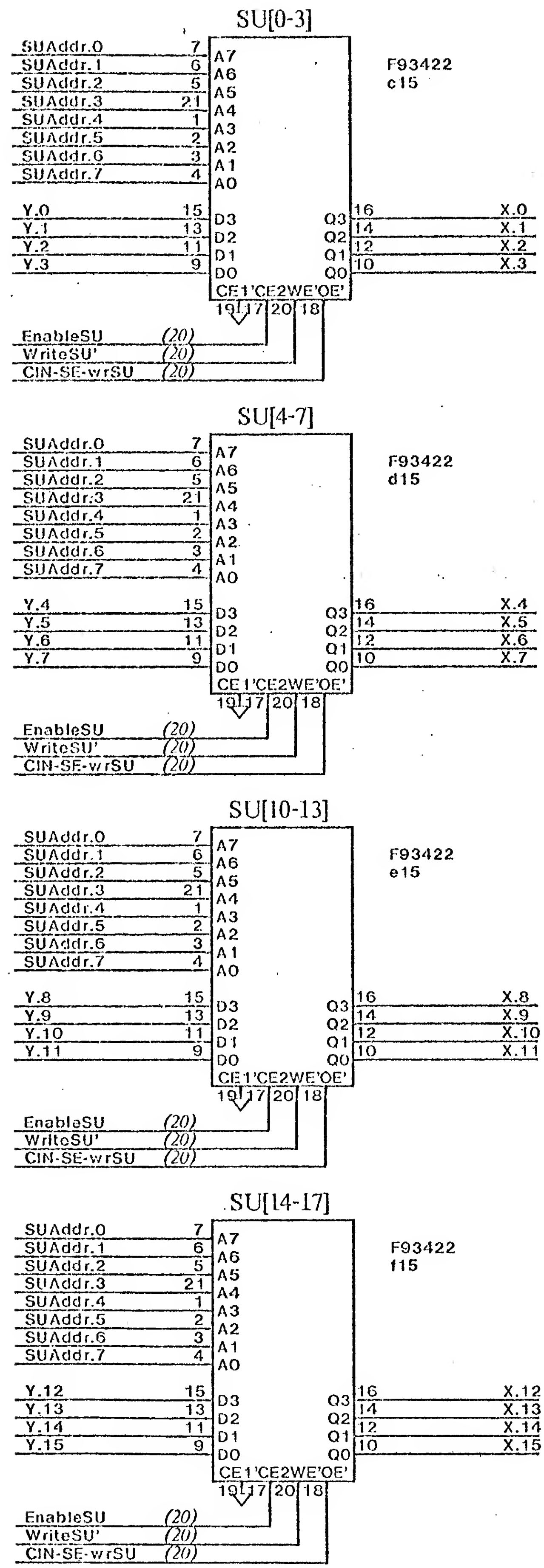
57[3] = 60 nS

SU write setup

5[1]	Data setup
39	WE

44[1] = 45 nS

F93422 data t-hold = 5 nS



RH & stackP X-bus disable

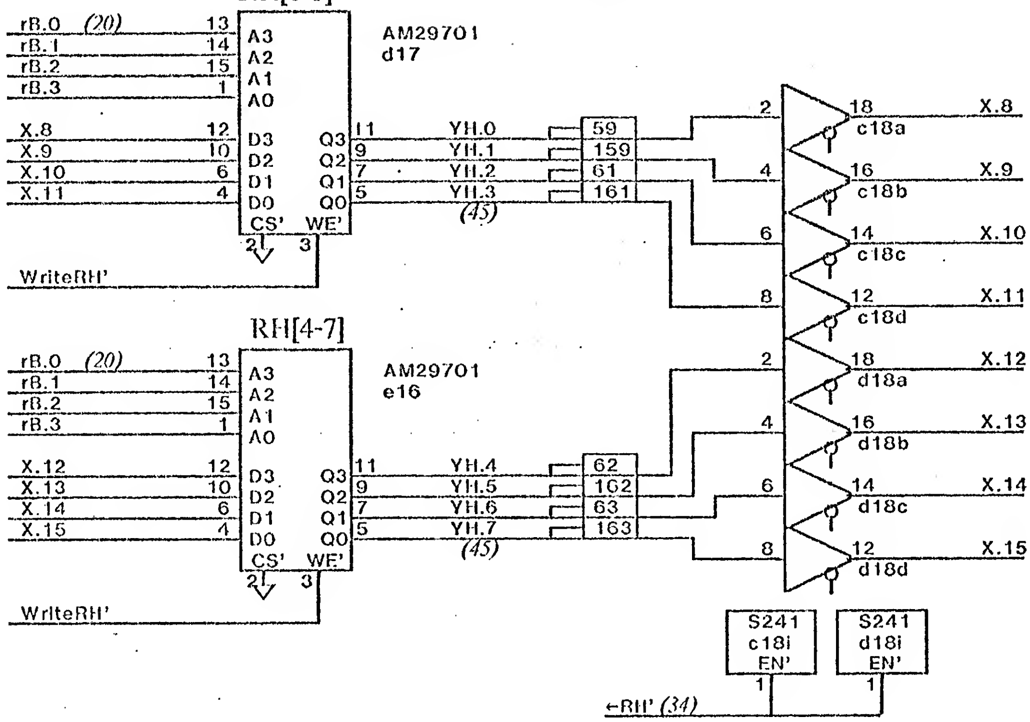
17[3]	↑ to fS
11[2]	S138 Enable to ←RH'
15	S241 En' to disabled X-bus
10	X-bus
53[5] = 58 nS	

XBus ← RH = max(64, 59) nS

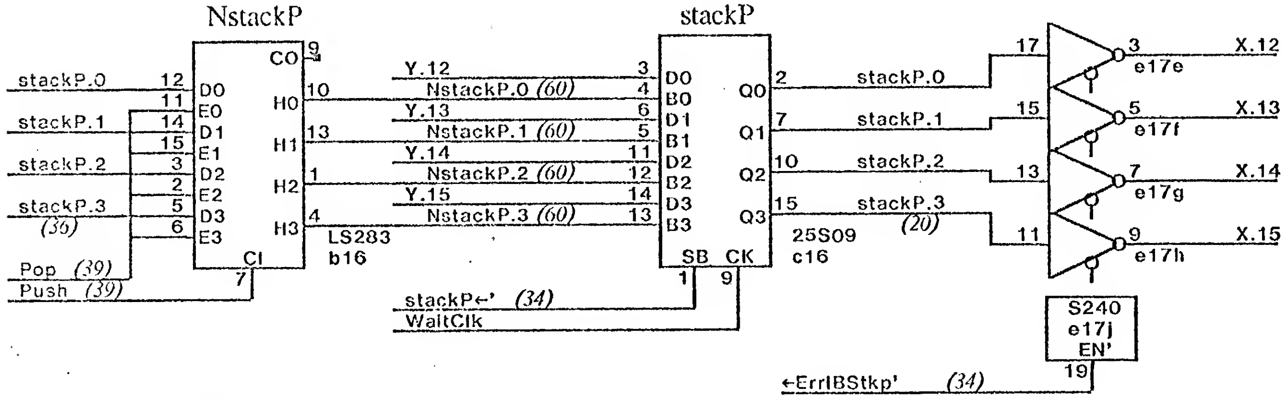
17[3]	↑ to rB'
25	tAA (write recovery = 20 nS)
9	YH to X
10	X-bus
61[3] = 64 nS	

34	↑ to ←RH'
15	S241 EN' to X-bus
10	X-bus
59 nS	

RH[0-3]



NstackP stackP

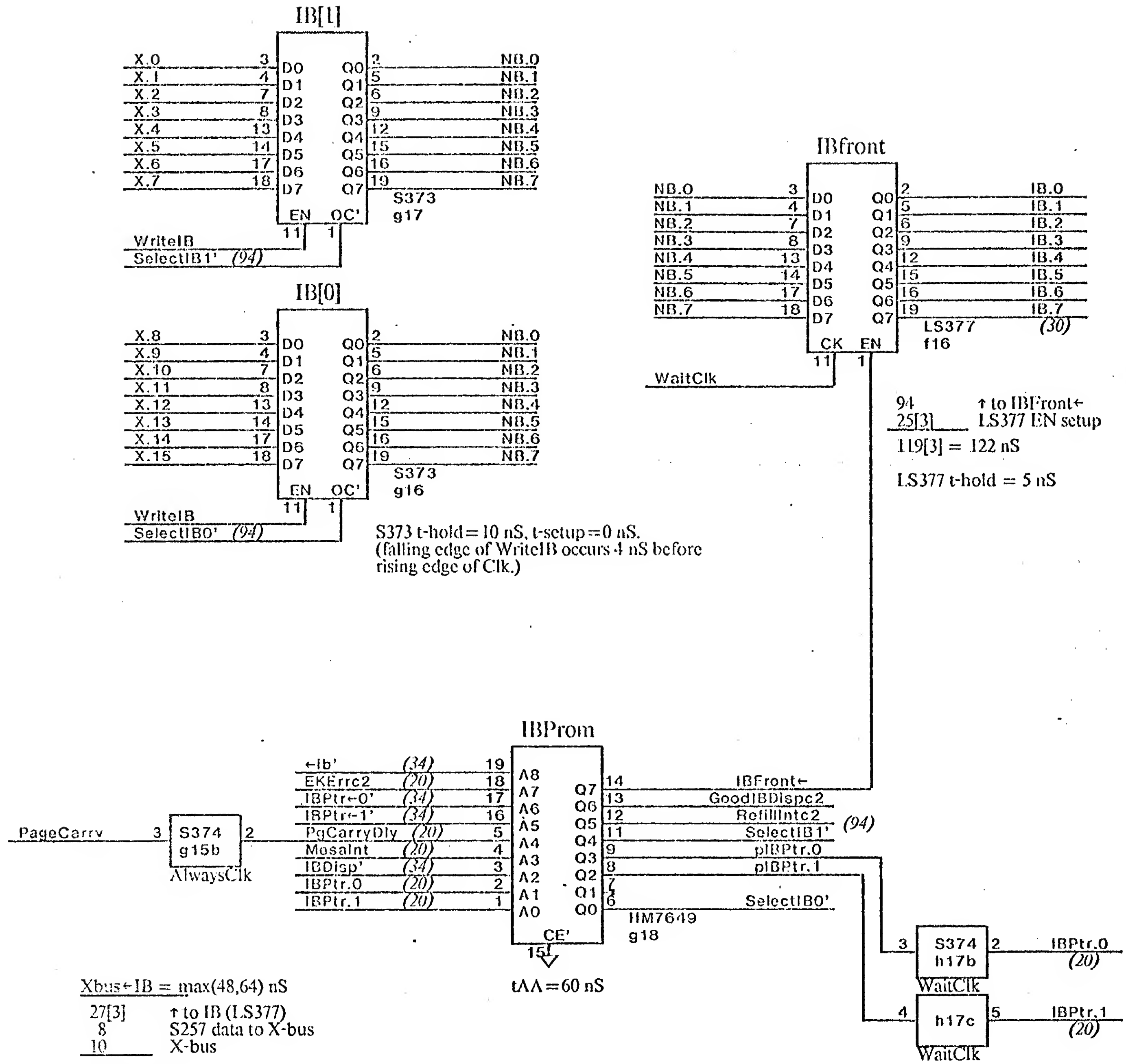


XBus ← stackP = max(59, 38) nS

17[3]	↑ to stackP
7	S240 data to X-bus
10	X-bus
34[3] = 38 nS	

34	↑ to ←ErrIntstackP'
15	S240 EN' to X-bus
10	X-bus
59 nS	

XEROX SDD	Project: Dandelion	RH, stackP	File: LionHead04.sil	Designer: Garner	Rev: C	Date: 10/30/79	Page: 04



Xbus←IB = max(48,64) nS

27[3] ↑ to IB (LS377)
8 S257 data to X-bus
10 X-bus
45[3] = 48 nS

33 ↑ to XBus←IB'
21 S257 E' to Xbus
10 X-bus
64 nS

IBFront ← Xbus = max(x + 36,88) nS

x Xbus to IB
13[1] S373 Data to NB
20[2] LS377 setup
33[3] = 36 nS

94 WriteIB rises
18[2] S373 EN to NB
20[2] LS377 setup
132[4] = 136 nS

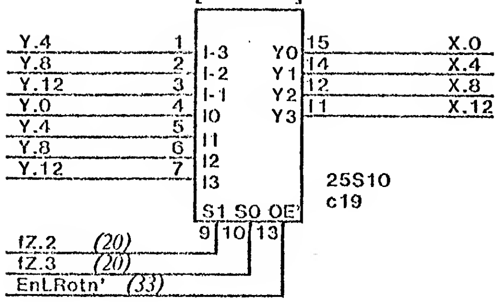
IBfront←IB[1]

34 ↑ to IBPtr←1'
60 tAA
18[2] SelectIB1' to NB
20[2] LS377 setup
132[4] = 136 nS

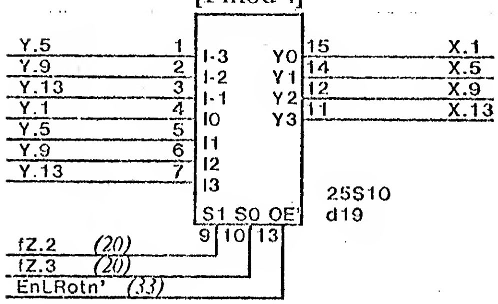
XEROX SDD	Project Dandelion	IB	File LionHead05.sil	Designer Garner	Rev B	Date 10/12/79	Page 05

LRotn

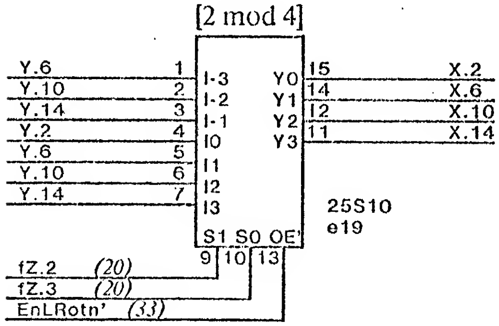
[0 mod 4]



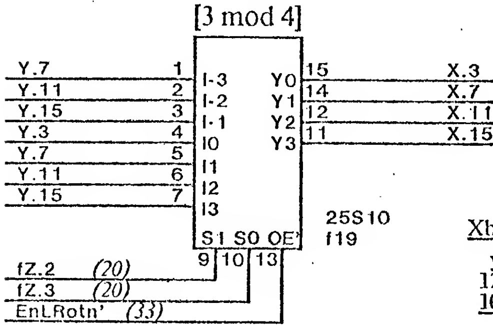
[1 mod 4]



[2 mod 4]



[3 mod 4]



fZ.2	fZ.3	
0	0	Left 0
0	1	Left 12
1	0	Left 8
1	1	Left 4

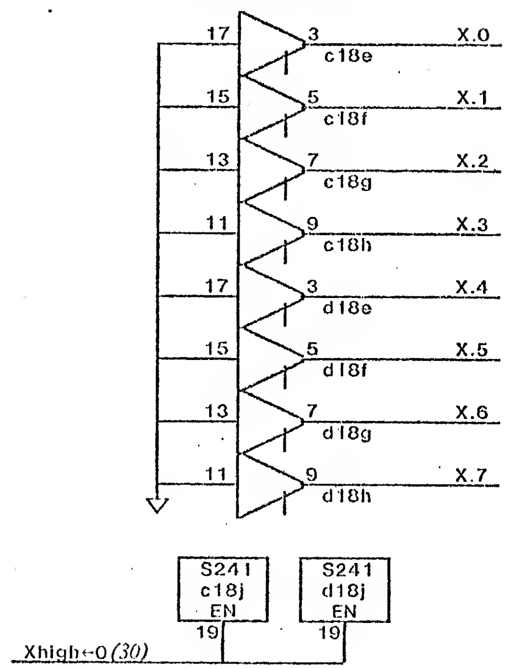
LRotn disable X-bus

33	↑ to EnLRotn'
15	25S10 OE' to X-bus
10	X-bus
58 nS	

Zero disable X-bus

30	↑ to Xhigh←0
15	S241 EN to X-bus
10	X-bus
55 nS	

Zero High XBus



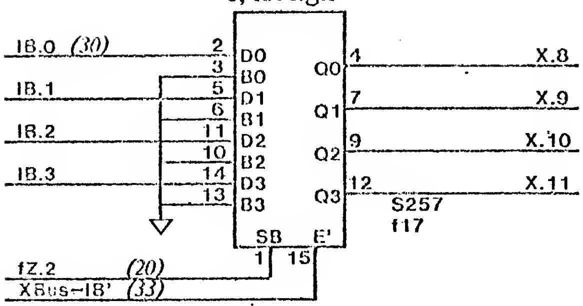
Xbus[0-7] ← 0

30	↑ to Xhigh←0
15	S241 OE
10	X-bus
55 nS	

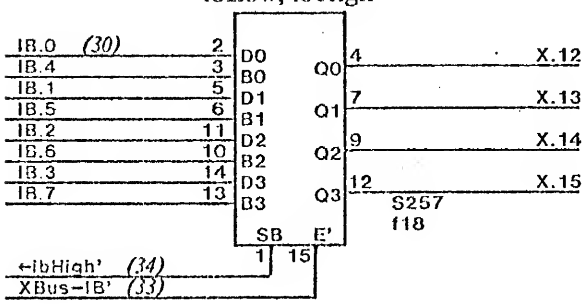
Xbus ← Y LRota = max (y+22, 56, 50) nS

y	↑ to Y bus
12	25S10 data in to out
10	X-bus
y +	22 nS
33	↑ to EnLRotn'
21	25S10 OE
10	X-bus
64 nS	
20	↑ to fZ.2
20	25S10 Select to X-bus
10	X-bus
50 nS	

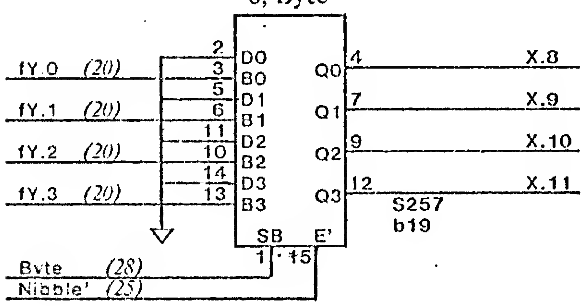
0, ibHigh



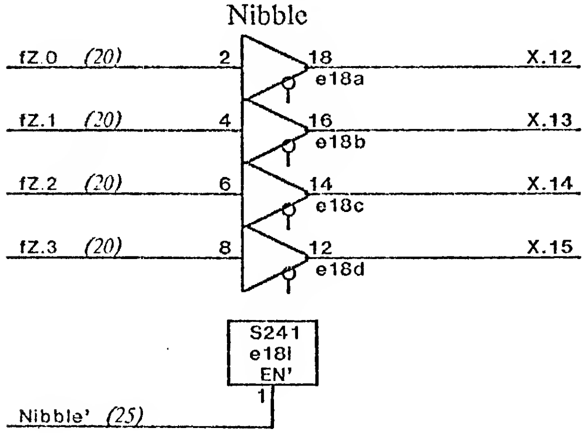
ibLow, ibHigh



0, Byte



Nibble



IB disable X-bus

33 ↑ to XBus←IB'
14 S257 E' to X-bus
10 X-bus
57 nS

Byte disable X-bus

25 ↑ to Nibble'
14 S257 E' to X-bus
10 X-bus
49 nS

Nibble disable X-bus

25 ↑ to Nibble'
15 S241 EN' to X-bus
10 X-bus
50 nS

Xbus ← Nibble = max(39, 50) nS

20 ↑ to fZ
9 S241 data to X-bus
10 X-bus
39 nS

25 ↑ to Nibble'
15 S241 EN' to X-bus
10 X-bus
50 nS

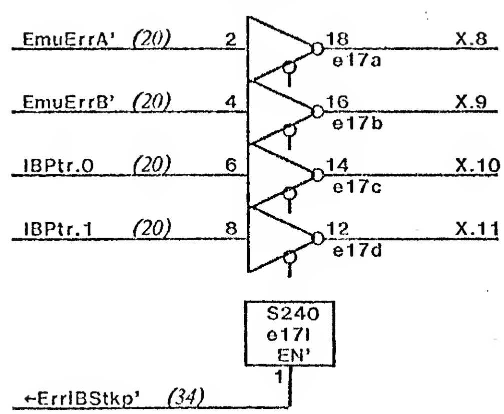
Xbus ← Byte = max(38, 56) nS

20 ↑ to fY
8 S257 data to X-bus
10 X-bus
38 nS

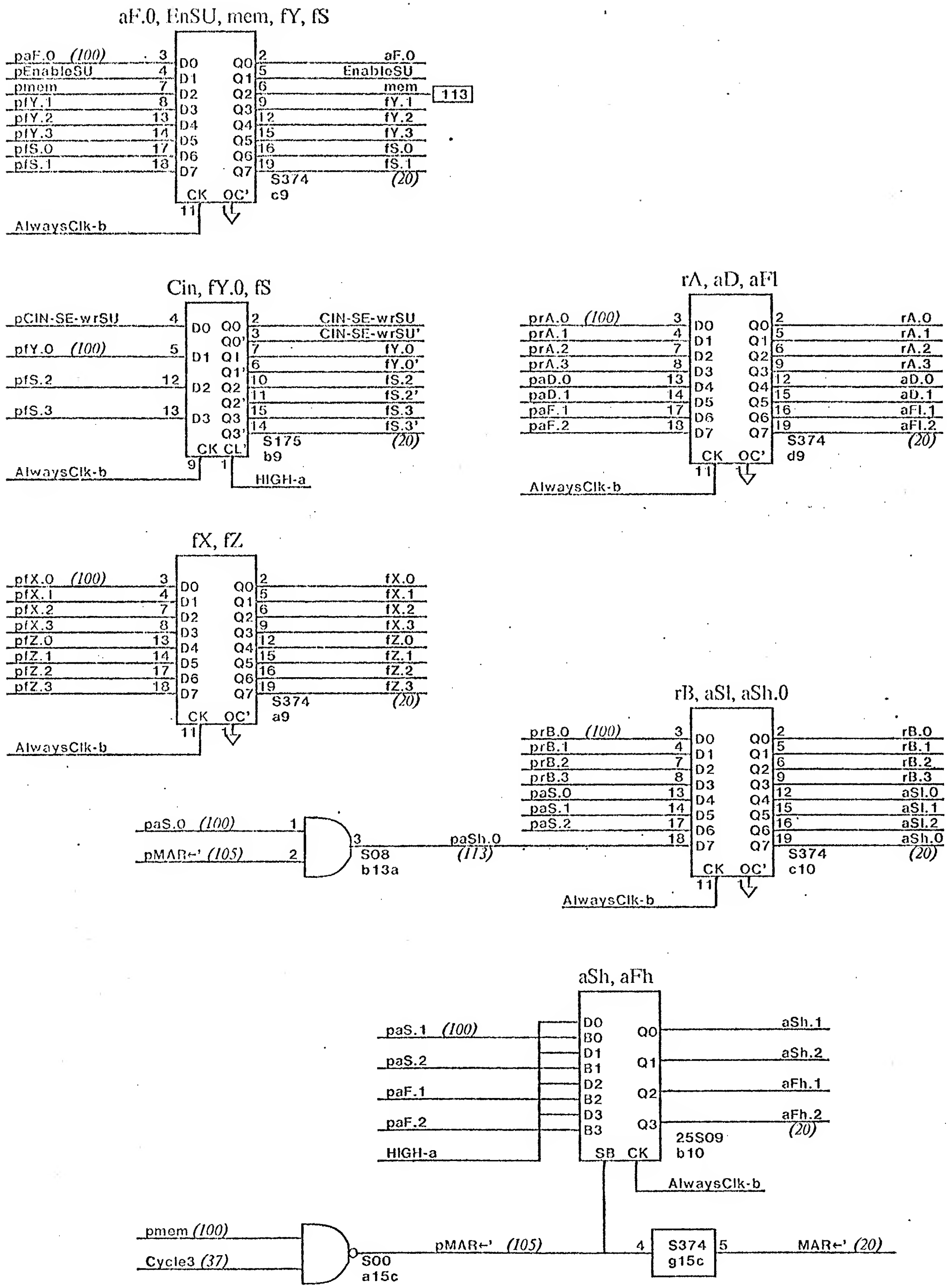
25 ↑ to Nibble'
21 S257 E' to X-bus
10 X-bus
56 nS

See stackP timings for ErrInt Status

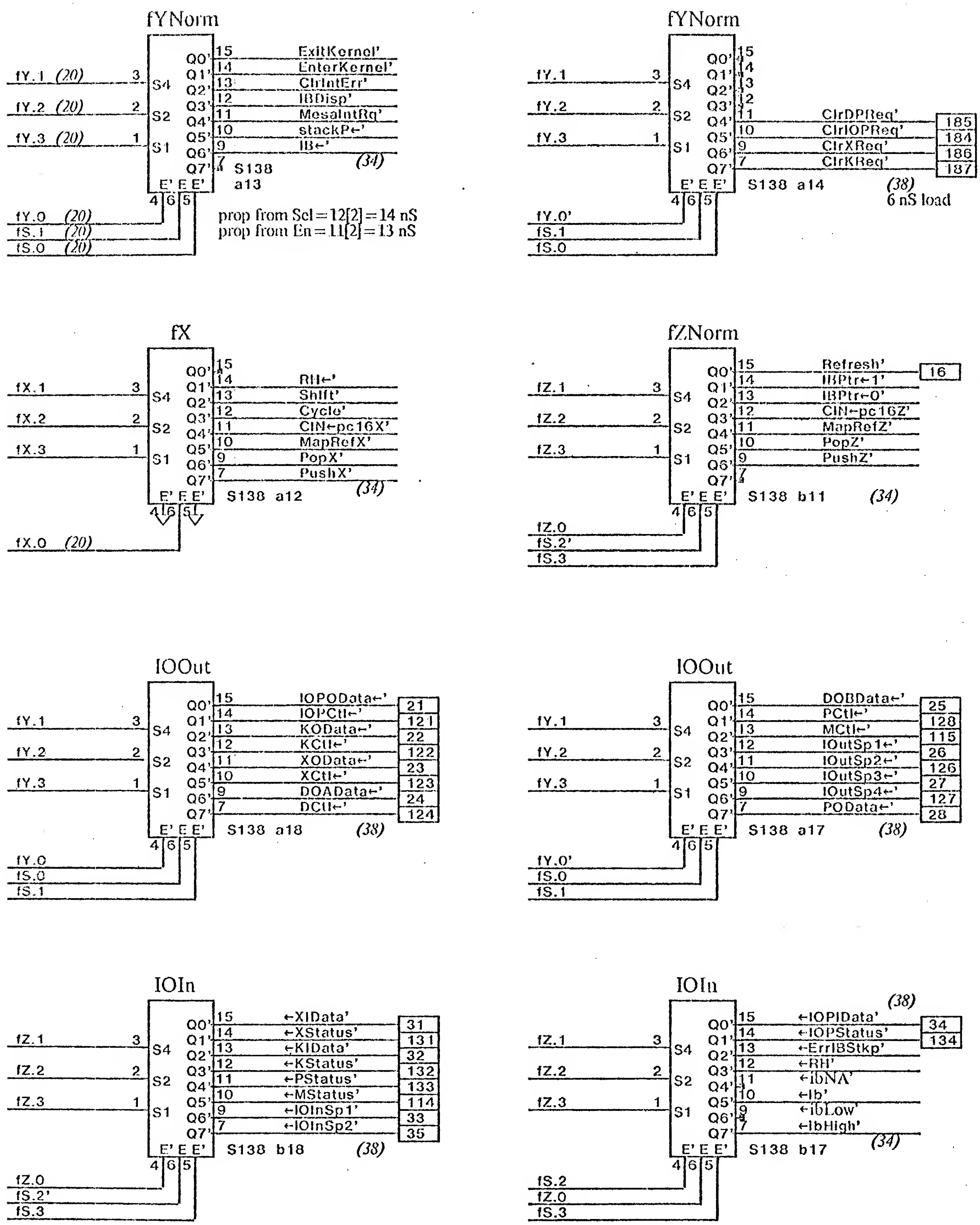
ErrIBPtr



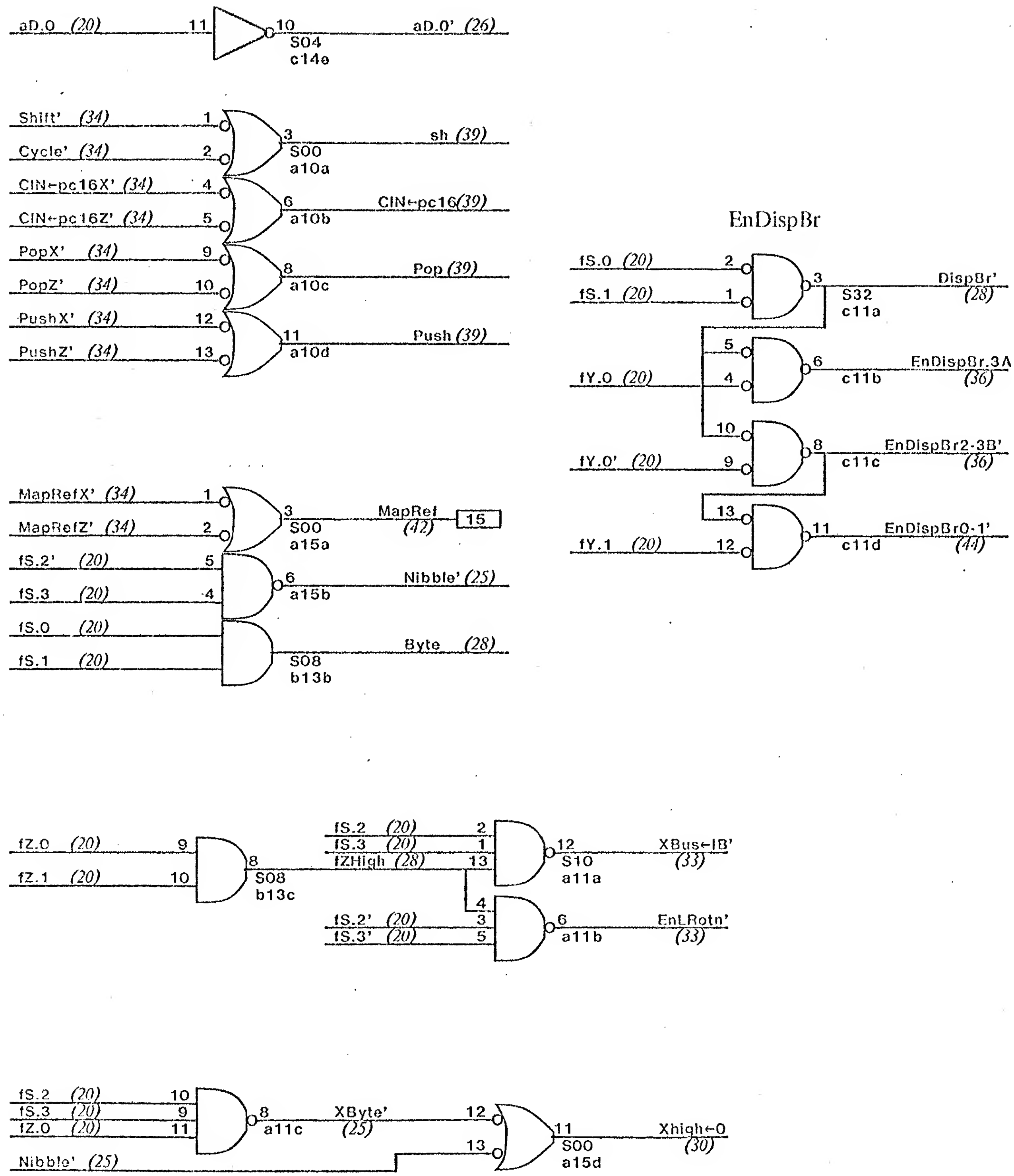
XEROX SDD	Project Dandelion	X Bus: IB, constants, ErrIntstackP	File LionHead07.sil	Designer Garner	Rev C	Date 10/30/79	Page 07



XEROX SDD	Project Dandelion	MIR	File LionHead08.sil	Designer Garner	Rev C	Date 10/30/79	Page 08



XEROX SDD	Project Dandelion	MIR Decoding I	File LionHead09.sil	Designer Garner	Rev B	Date 10/12/79	Page 09



XEROX SDD	Project Dandelion	MIR Decoding II	File LionHead10.sil	Designer Garner	Rev C	Date 10/30/79	Page 10

DispBr[2-3] = max(c+26,55,68)

20	↑ to fY
15[2]	S151/258 select to DispBr
5	S00 in to pTC
6[1]	S64 in to pNIA
5[1]	25S09/S374 setup

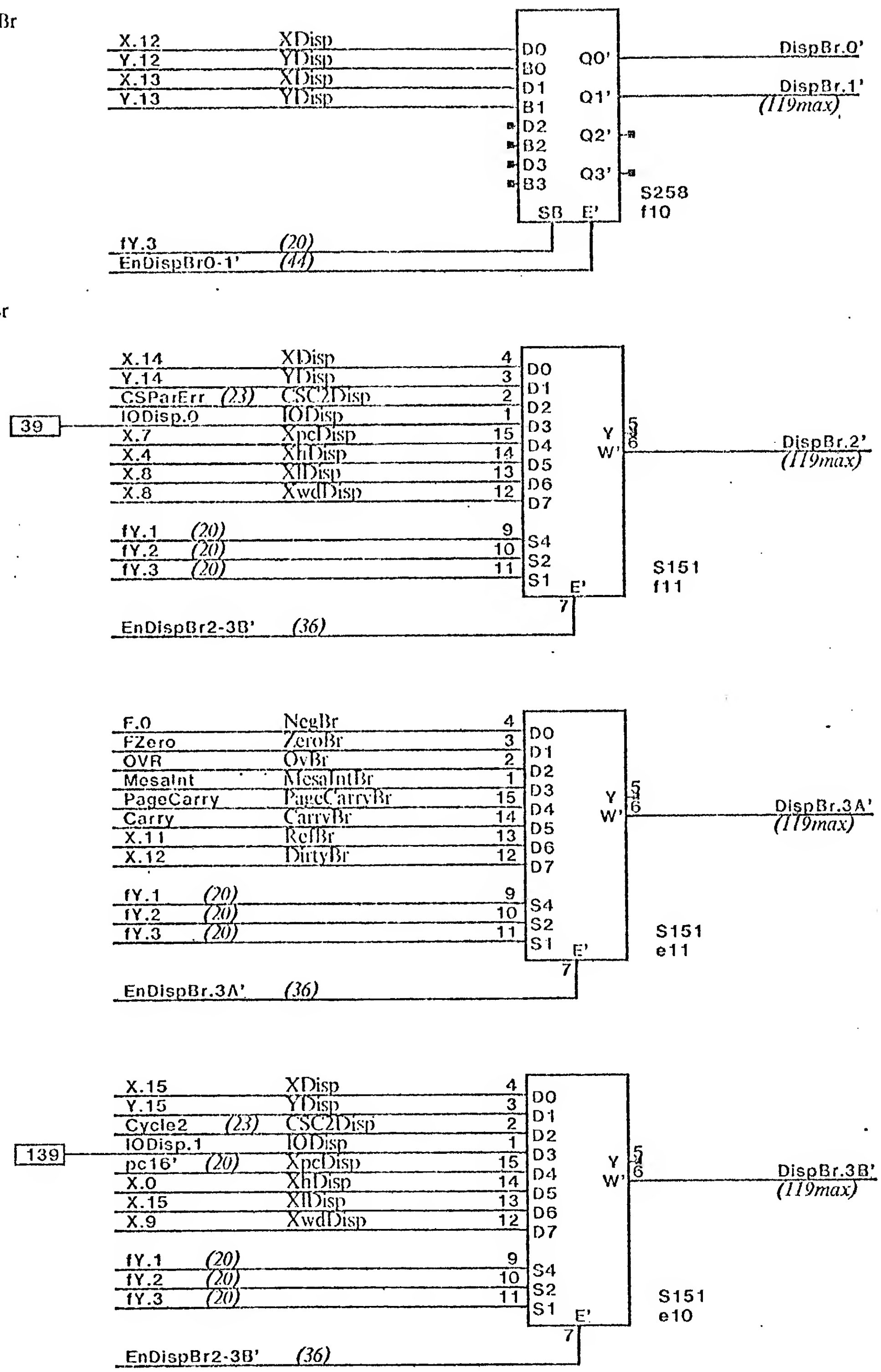
51[4] = 55 nS

36	↑ to EnDispBr'
13[2]	S151/258 E' to DispBr
5	S00 in to pTC
6[1]	S64 in to pNIA
5[1]	25S09/S374 setup

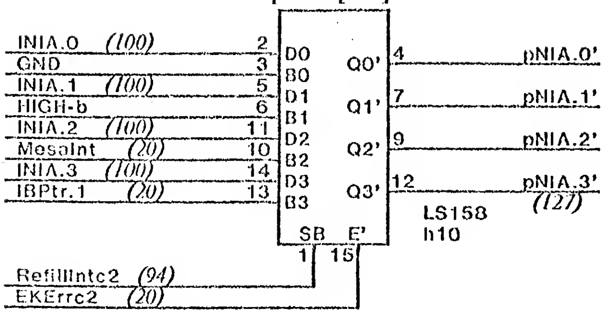
65[4] = 68 nS

c	condition source
7[1]	S151/258 data to DispBr
5	S00 in to pTC
6[1]	S64 in to pNIA
5[1]	25S09/S374 setup

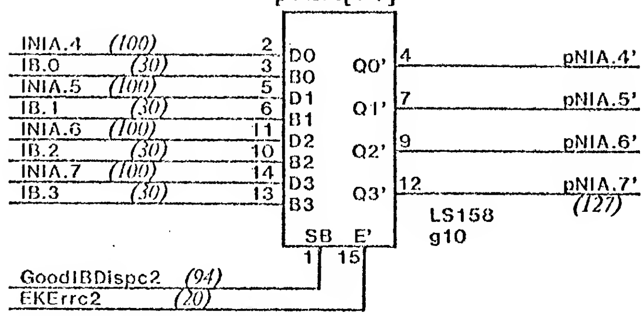
c+23[3] = c+26 nS



pNIA[0-3]



pNIA[4-7]

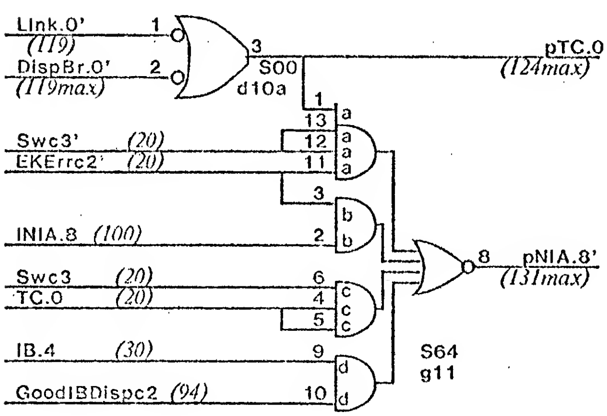


pNIA[0-7]=max(127, 120, 46) nS

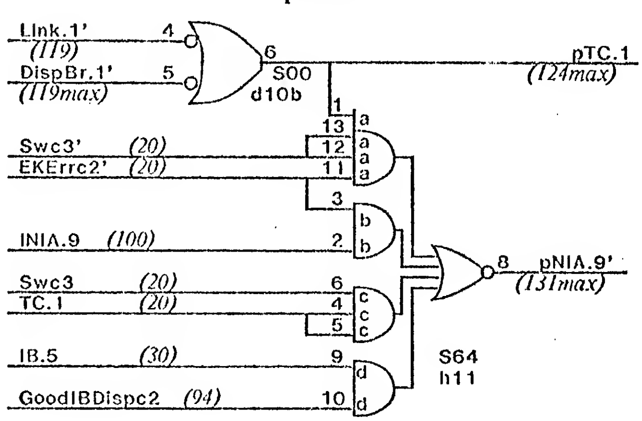
94	↑ to RefillIntc2	100	↑ to INIA	20	↑ to EKErrc2
24[3]	LS158 SB to pNIA'	12[2]	LS158 data to pNIA'	18[2]	LS158 E' to pNIA'
5[1]	25S09/S374 setup	5[1]	25S09/S374 setup	5[1]	25S09/S374 setup
123[4]=127 nS		117[3]=120 nS		43[3]=46nS	

(See page 11 for pNIA[8-11] timing)

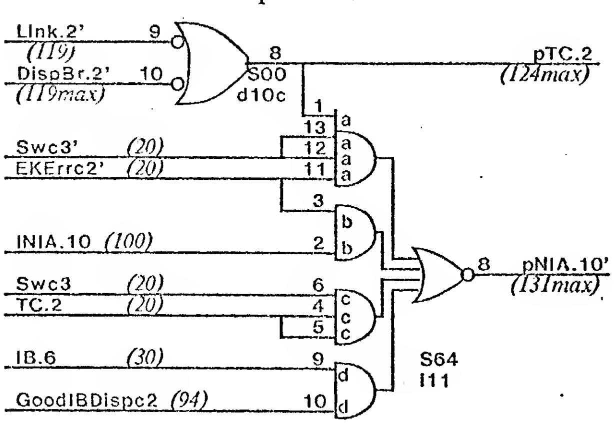
pNIA.8



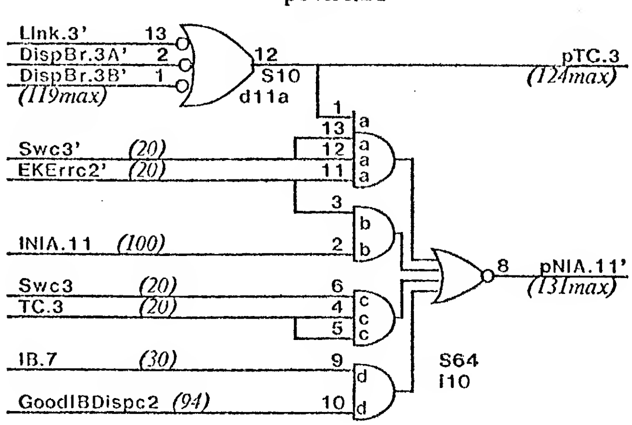
pNIA.9



pNIA.10



pNIA.11



CS Timing

17 ↑ to NIA'
13 transmission delay
70 tAA 2147L
100 nS

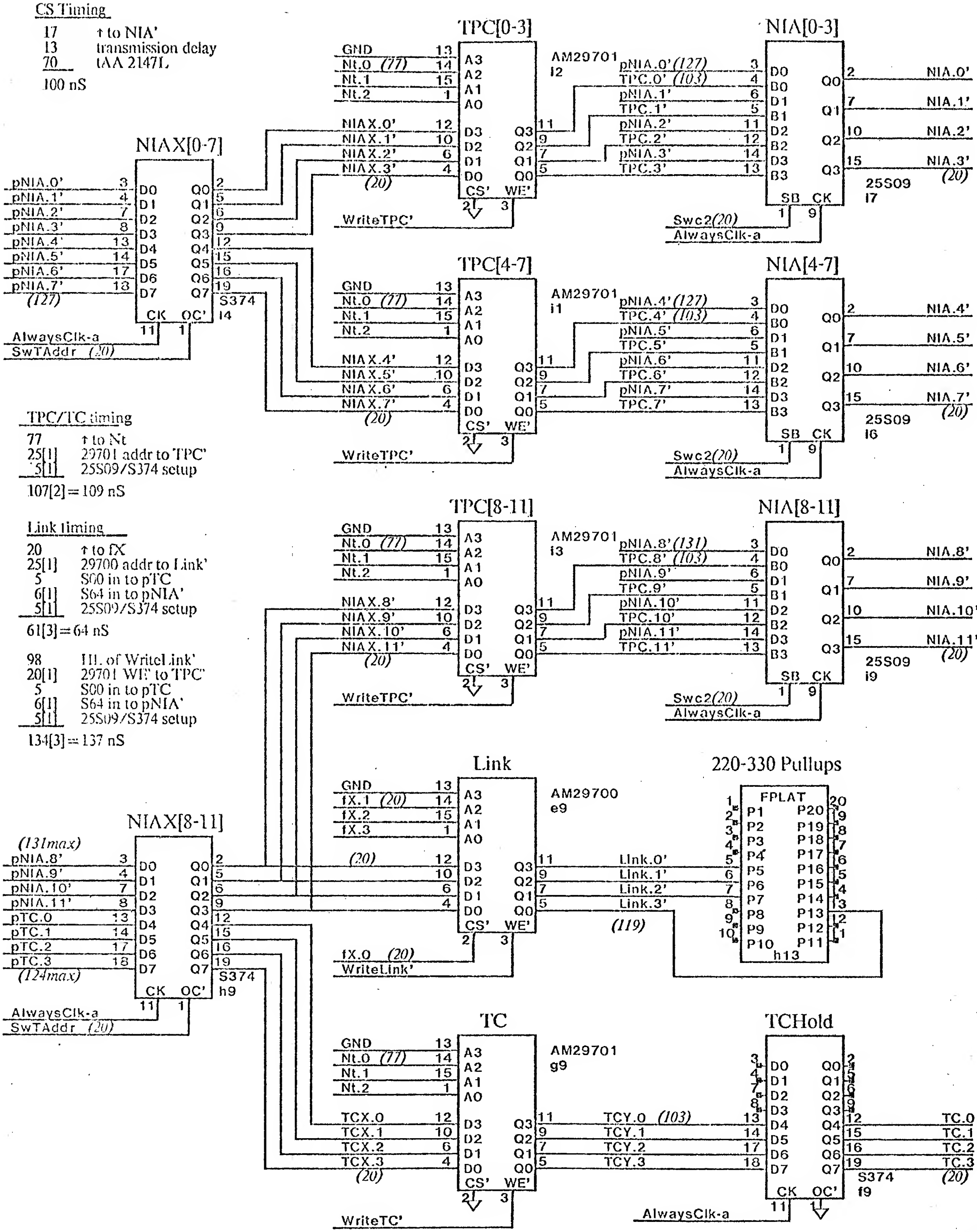
TPC/TC timing

77 ↑ to Nt
25[1] 29701 addr to TPC'
5[1] 25S09/S374 setup
107[2] = 109 nS

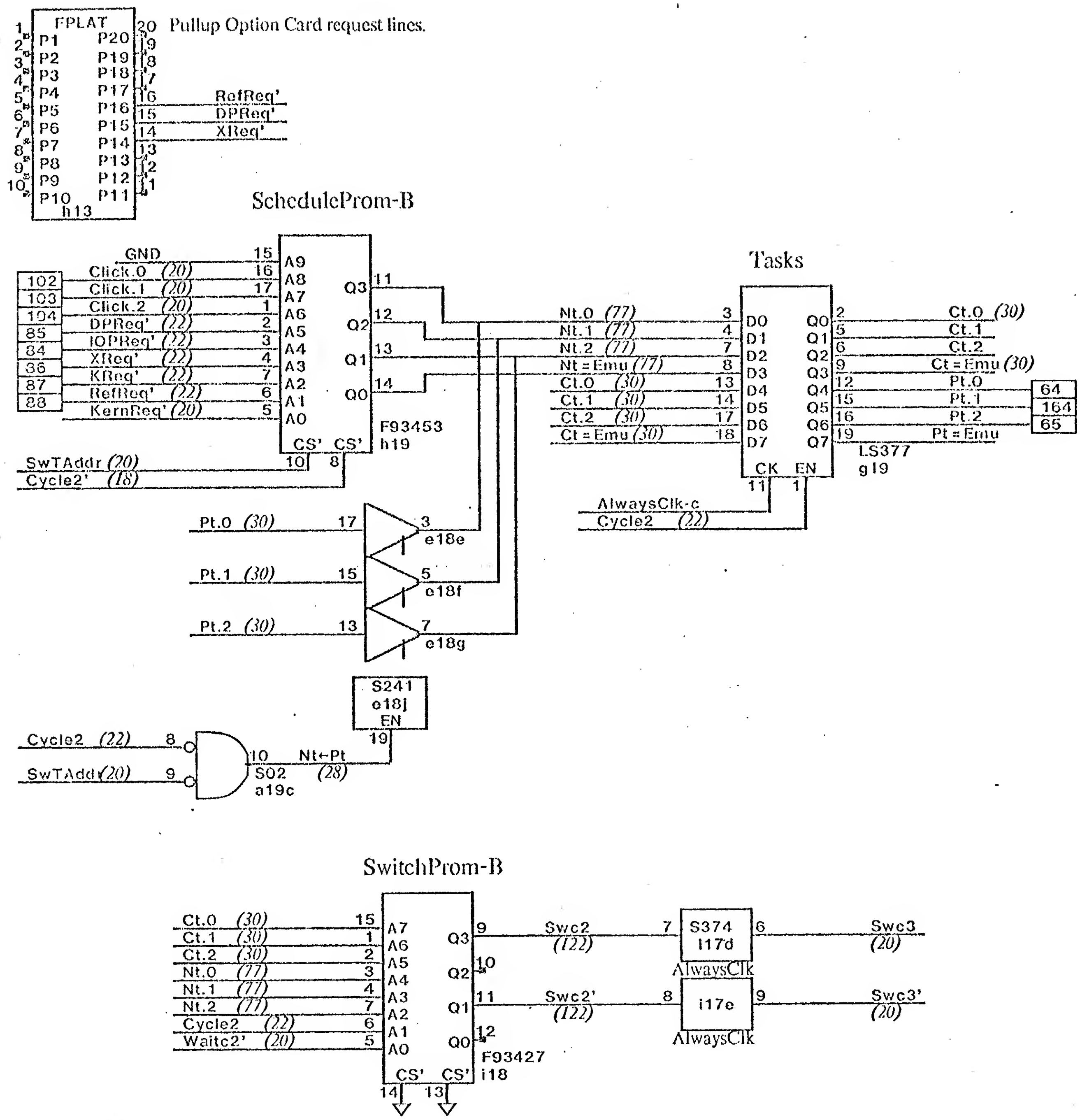
Link timing

20 ↑ to fX
25[1] 29700 addr to Link'
5 S00 in to pTC
6[1] S64 in to pNIA'
5[1] 25S09/S374 setup
61[3] = 64 nS

98 HL of WriteLink'
20[1] 29701 WE to TPC'
5 S00 in to pTC
6[1] S64 in to pNIA'
5[1] 25S09/S374 setup
134[3] = 137 nS



XEROX SDD	Project Dandelion	TPC, TC, & Link	File LionHead13.sil	Designer Garner	Rev B	Date 10/10/79	Page 13



	Nt (Prom)	Nt	Ct	Pt
c1	3-S	Previous	Current	Previous
c2	Next	Next	Current	Previous
c3	3-S	Current	Next	Current

Swc2 timing = max(133,101,101)

22	↑ to Kreq'
55	F93453 addr to Nt
45	F93427 addr to Swc2
10[1]	25S09 SB setup

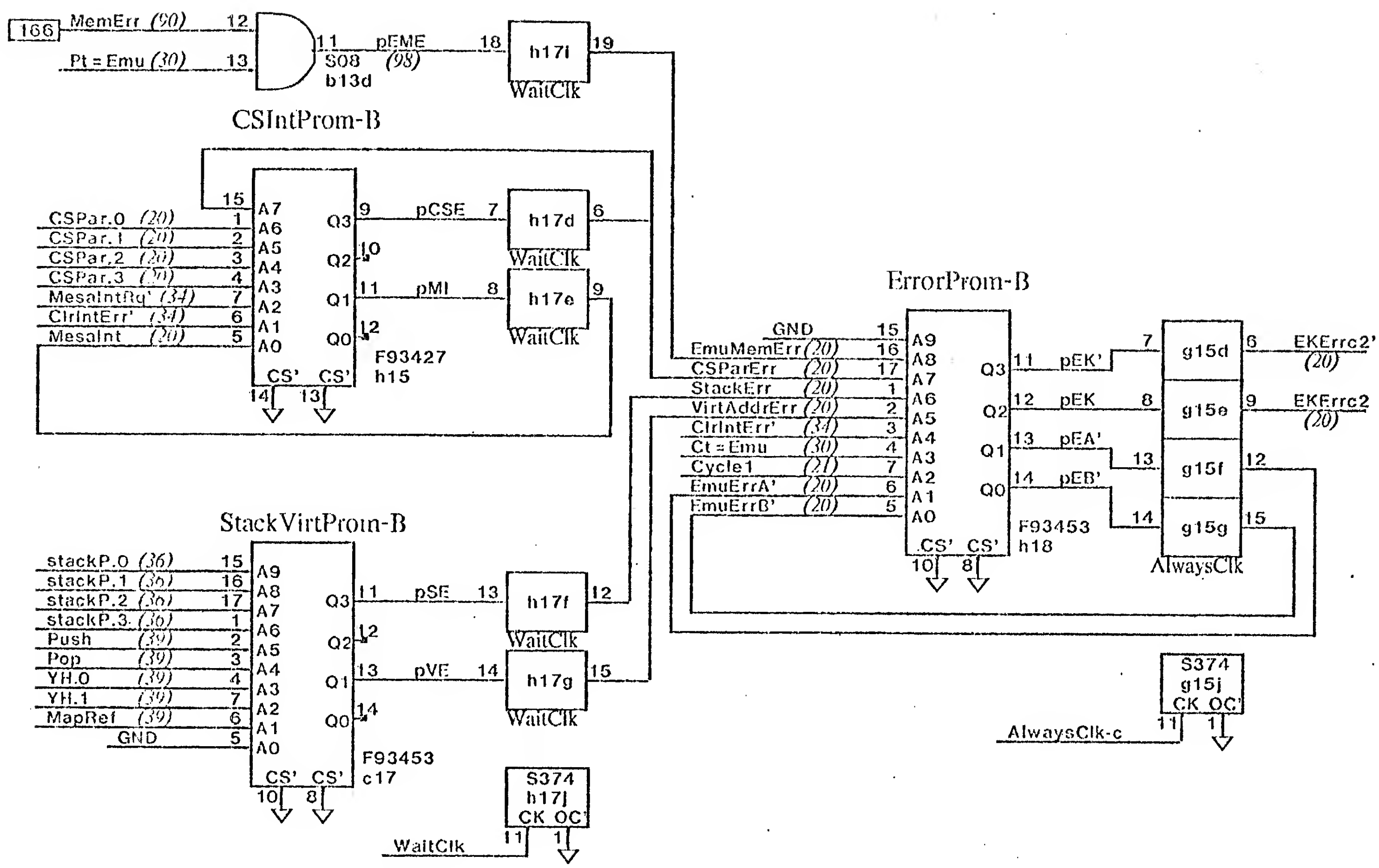
132[1] = 133 nS

20	↑ to SwTAddr
25	F93453 CS' to Nt
45	F93427 addr to Swc2
10[1]	25S09 SB setup

100[1] = 101 nS

28	↑ to Nt←Pt
15[2]	S241 EN to Nt
45	F93427 addr to Swc2
10[1]	25S09 SB setup

98[3] = 101 nS



CSIntProm/KernPC16 timing

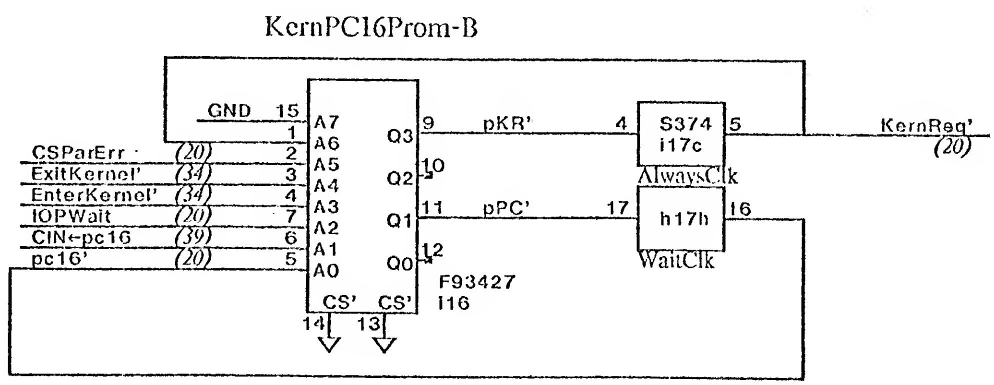
34	↑ to ClrIntErr'
45	93427 addr to pMI
5[1]	S374 setup

84[1]=85 nS

StackVirtProm/ErrorProm timing

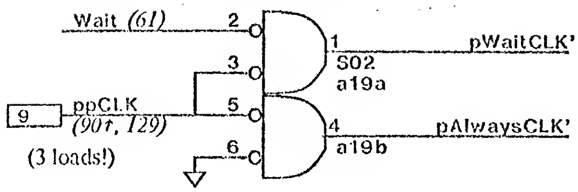
39	↑ to YH.0
55	93453 addr to pVE
5[1]	S374 setup

89[1]=90 nS

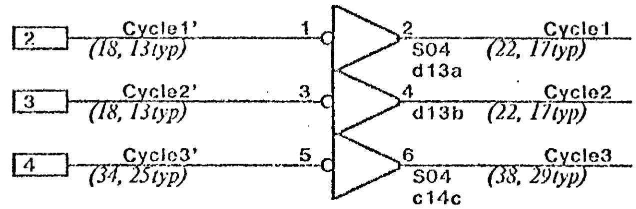


XEROX SDD	Project: Dandelion	Error, Emulator, & Kernel Proms	File: LionHead15.sil	Designer: Garner	Rev: B	Date: 10/12/79	Page: 15

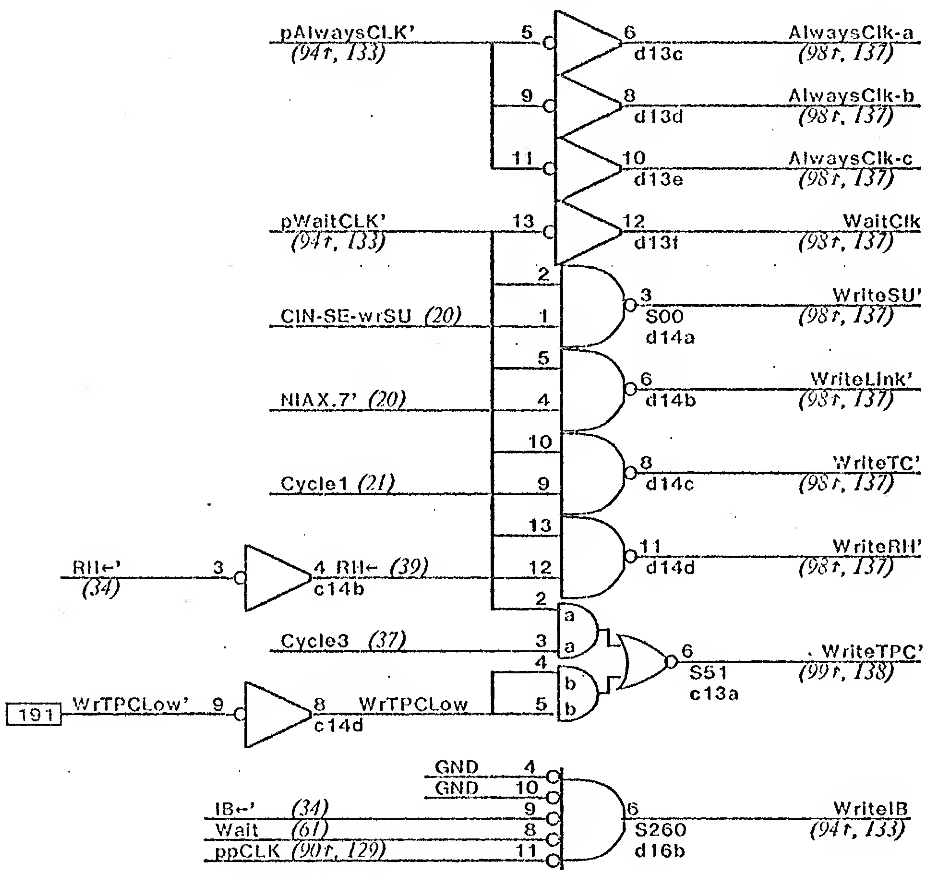
Clock Receivers



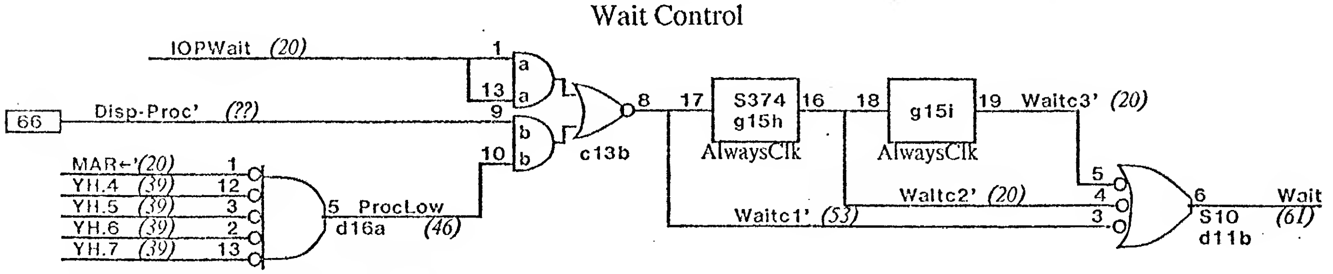
Cycles



Qualified Clocks

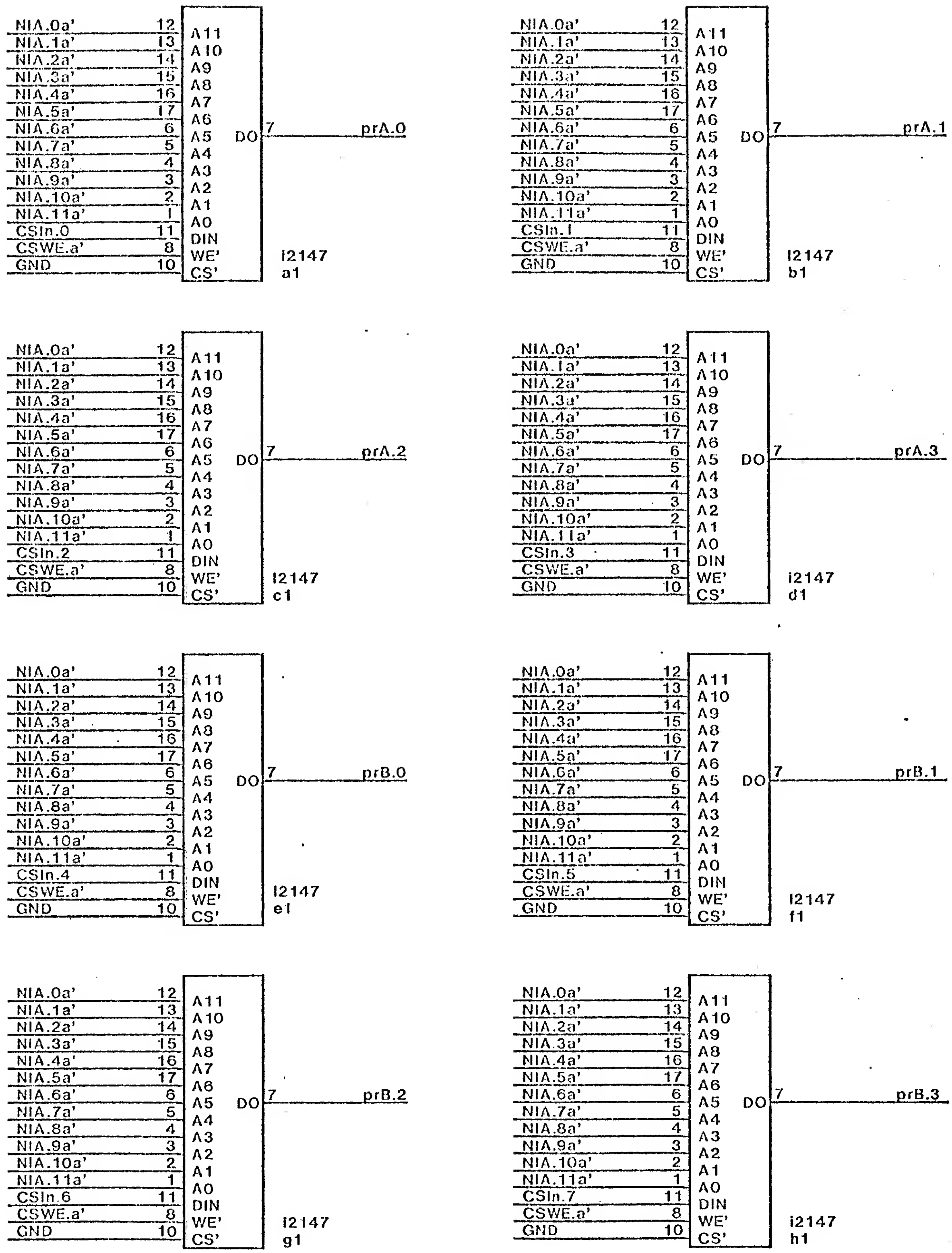


Wait Control

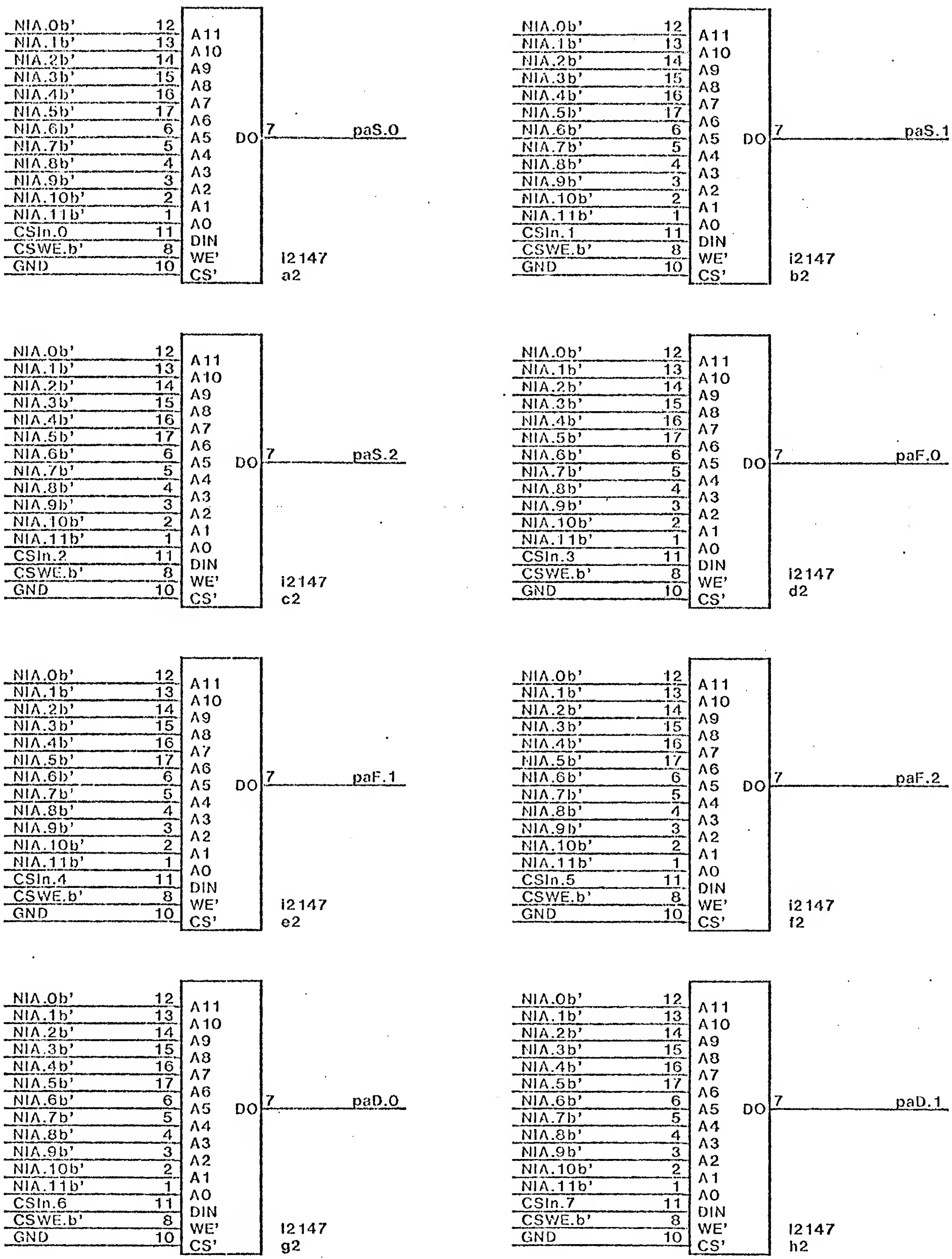


Detects low bank for max 1024K mem.

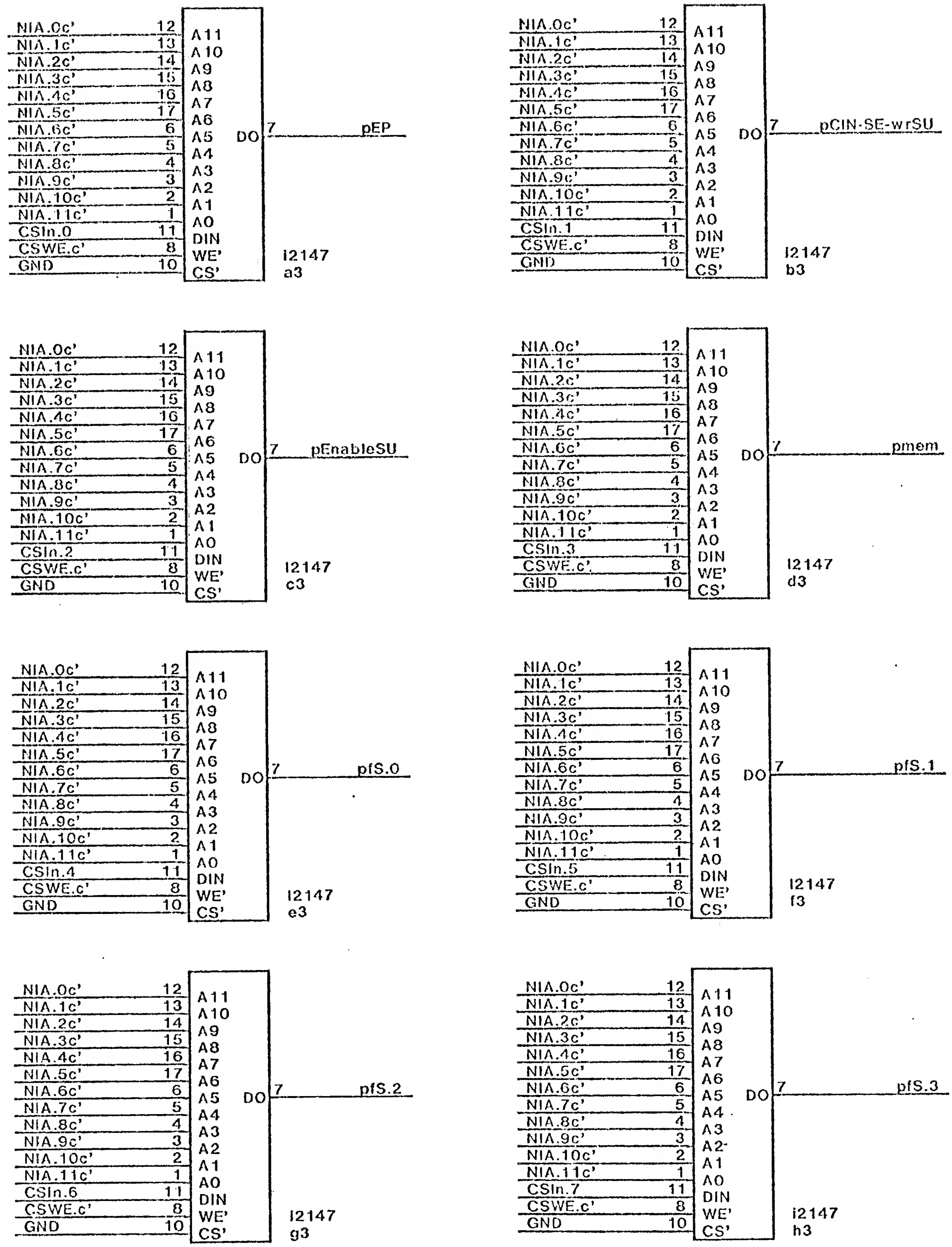
XEROX SDD	Project		File	Designer	Rev	Date	Page
	Dandelion	Clocks, Wait	LionHead16.sil	Garner	B	10/10/79	16



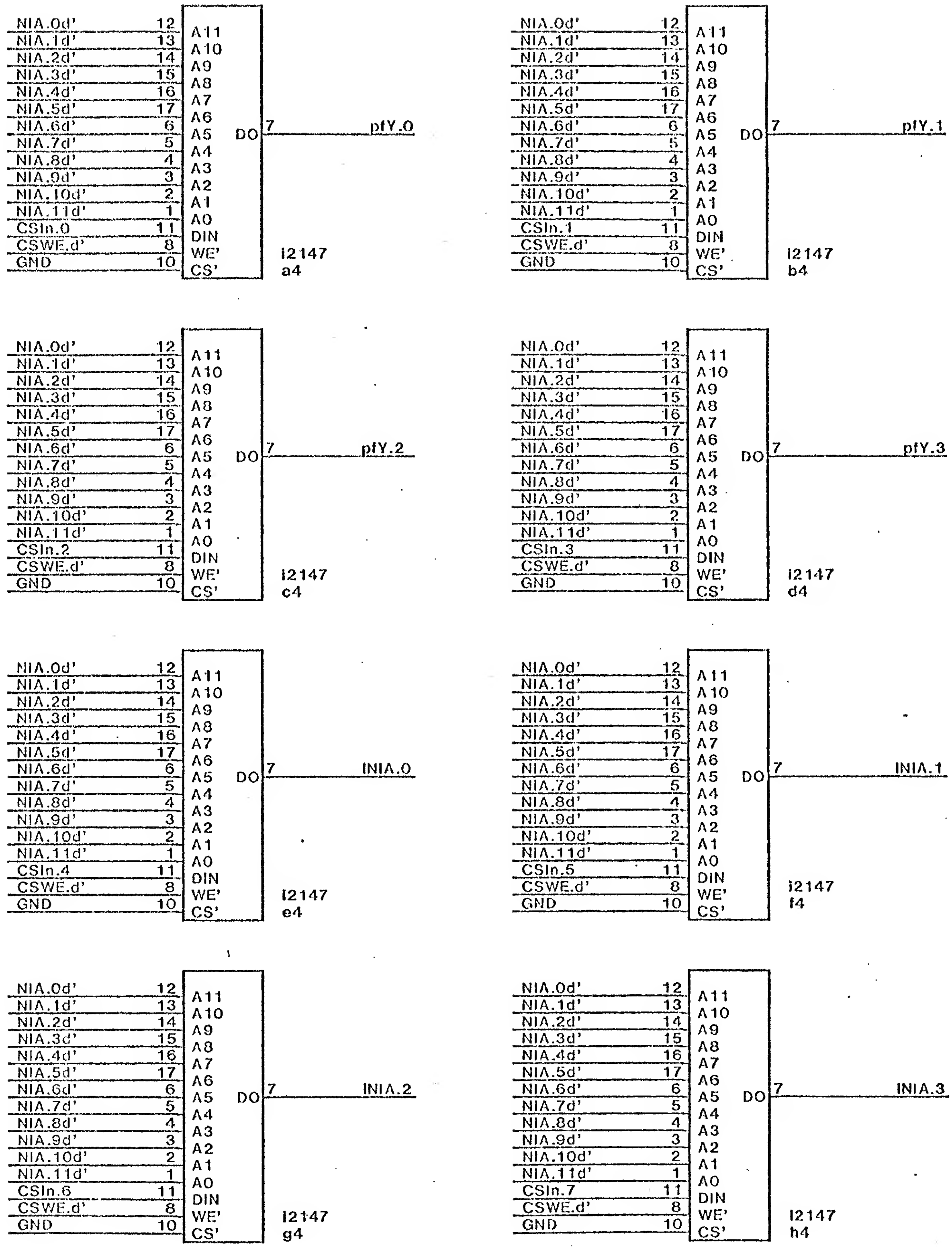
XEROX SDD	Project		File	Designer	Rev	Date	Page
	Dandelion	Control Store A [0-7]	LionHead17.sil	Garner	B	10/10/79	17

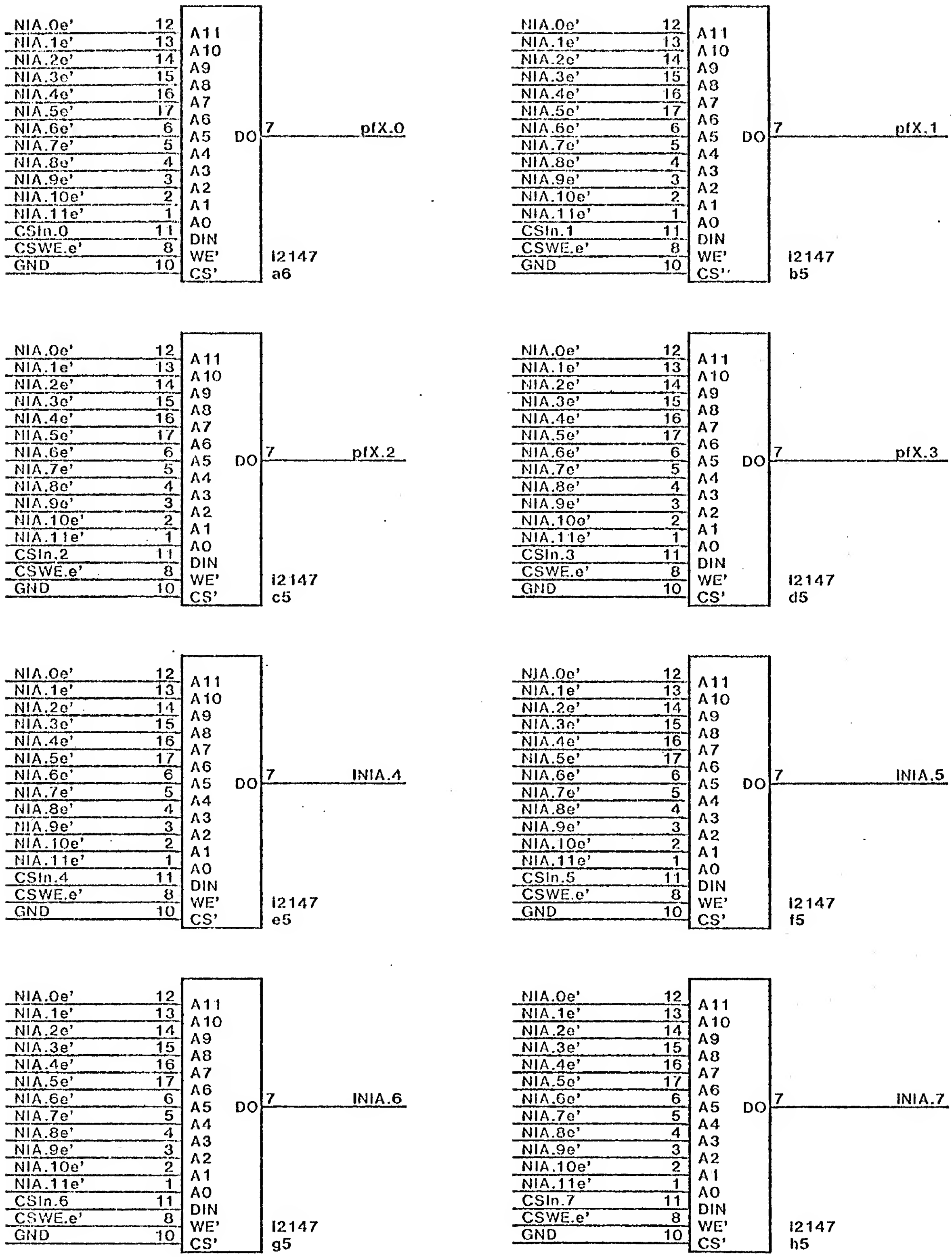


XEROX SDD	Project Dandelion	Control Store B [8-15]	File LionHead18.sil	Designer Garner	Rev B	Date 10/10/79	Page 18

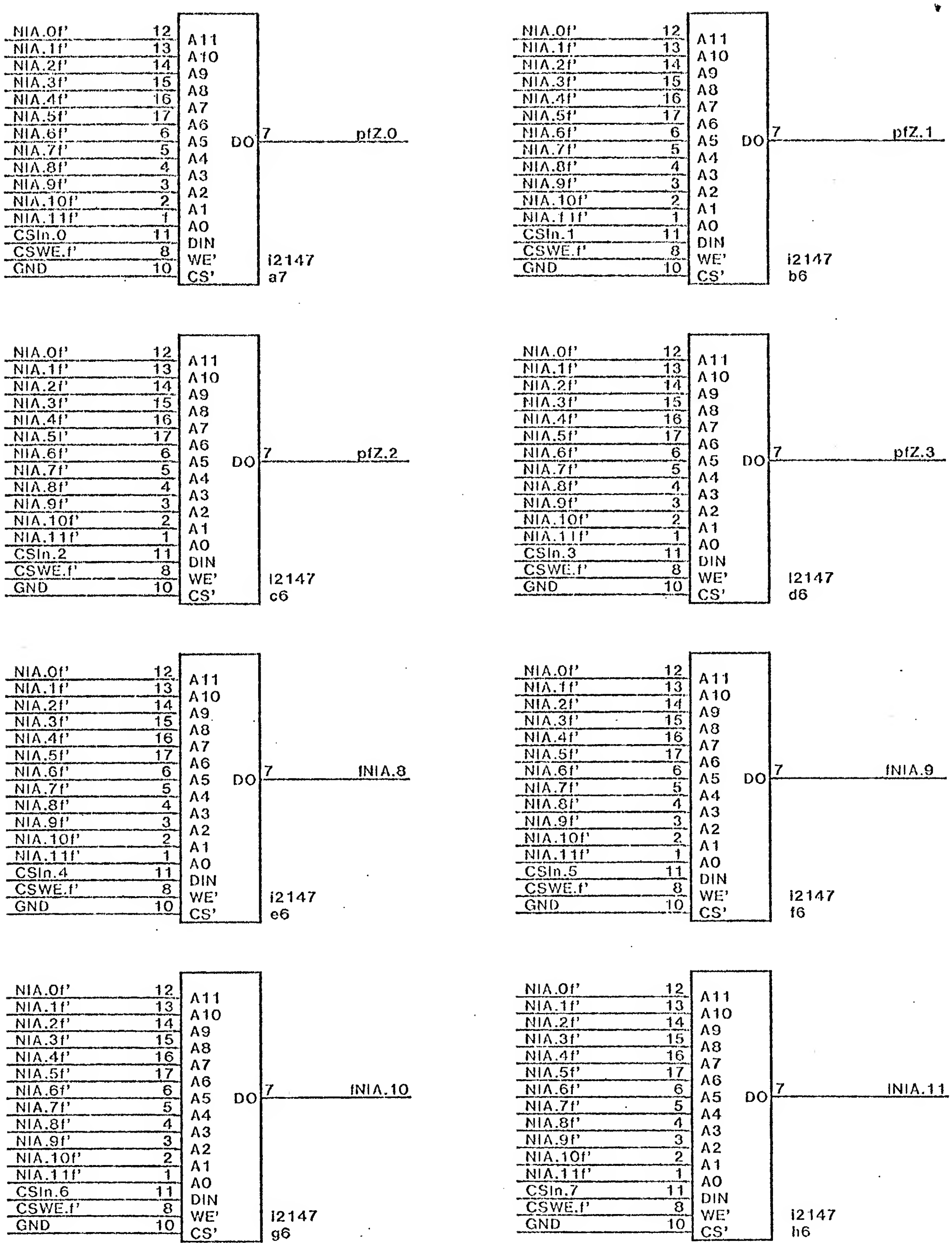


NIA.0c' 12 A11
NIA.1c' 13 A10
NIA.2c' 14 A9
NIA.3c' 15 A8
NIA.4c' 16 A7
NIA.5c' 17 A6
NIA.6c' 6 A5
NIA.7c' 5 A4
NIA.8c' 4 A3
NIA.9c' 3 A2
NIA.10c' 2 A1
NIA.11c' 1 A0
CSIn.0 11 DIN
CSWE.c' 8 WE'
GND 10 CS'
DO 7 pEP
i2147
a3
NIA.0c' 12 A11
NIA.1c' 13 A10
NIA.2c' 14 A9
NIA.3c' 15 A8
NIA.4c' 16 A7
NIA.5c' 17 A6
NIA.6c' 6 A5
NIA.7c' 5 A4
NIA.8c' 4 A3
NIA.9c' 3 A2
NIA.10c' 2 A1
NIA.11c' 1 A0
CSIn.1 11 DIN
CSWE.c' 8 WE'
GND 10 CS'
DO 7 pCIN-SE-wrSU
i2147
b3
NIA.0c' 12 A11
NIA.1c' 13 A10
NIA.2c' 14 A9
NIA.3c' 15 A8
NIA.4c' 16 A7
NIA.5c' 17 A6
NIA.6c' 6 A5
NIA.7c' 5 A4
NIA.8c' 4 A3
NIA.9c' 3 A2
NIA.10c' 2 A1
NIA.11c' 1 A0
CSIn.2 11 DIN
CSWE.c' 8 WE'
GND 10 CS'
DO 7 pEnableSU
i2147
c3
NIA.0c' 12 A11
NIA.1c' 13 A10
NIA.2c' 14 A9
NIA.3c' 15 A8
NIA.4c' 16 A7
NIA.5c' 17 A6
NIA.6c' 6 A5
NIA.7c' 5 A4
NIA.8c' 4 A3
NIA.9c' 3 A2
NIA.10c' 2 A1
NIA.11c' 1 A0
CSIn.3 11 DIN
CSWE.c' 8 WE'
GND 10 CS'
DO 7 pmem
i2147
d3
NIA.0c' 12 A11
NIA.1c' 13 A10
NIA.2c' 14 A9
NIA.3c' 15 A8
NIA.4c' 16 A7
NIA.5c' 17 A6
NIA.6c' 6 A5
NIA.7c' 5 A4
NIA.8c' 4 A3
NIA.9c' 3 A2
NIA.10c' 2 A1
NIA.11c' 1 A0
CSIn.4 11 DIN
CSWE.c' 8 WE'
GND 10 CS'
DO 7 pfS.0
i2147
e3
NIA.0c' 12 A11
NIA.1c' 13 A10
NIA.2c' 14 A9
NIA.3c' 15 A8
NIA.4c' 16 A7
NIA.5c' 17 A6
NIA.6c' 6 A5
NIA.7c' 5 A4
NIA.8c' 4 A3
NIA.9c' 3 A2
NIA.10c' 2 A1
NIA.11c' 1 A0
CSIn.5 11 DIN
CSWE.c' 8 WE'
GND 10 CS'
DO 7 pfS.1
i2147
f3
NIA.0c' 12 A11
NIA.1c' 13 A10
NIA.2c' 14 A9
NIA.3c' 15 A8
NIA.4c' 16 A7
NIA.5c' 17 A6
NIA.6c' 6 A5
NIA.7c' 5 A4
NIA.8c' 4 A3
NIA.9c' 3 A2
NIA.10c' 2 A1
NIA.11c' 1 A0
CSIn.6 11 DIN
CSWE.c' 8 WE'
GND 10 CS'
DO 7 pfS.2
i2147
g3
NIA.0c' 12 A11
NIA.1c' 13 A10
NIA.2c' 14 A9
NIA.3c' 15 A8
NIA.4c' 16 A7
NIA.5c' 17 A6
NIA.6c' 6 A5
NIA.7c' 5 A4
NIA.8c' 4 A3
NIA.9c' 3 A2
NIA.10c' 2 A1
NIA.11c' 1 A0
CSIn.7 11 DIN
CSWE.c' 8 WE'
GND 10 CS'
DO 7 pfS.3
i2147
h3

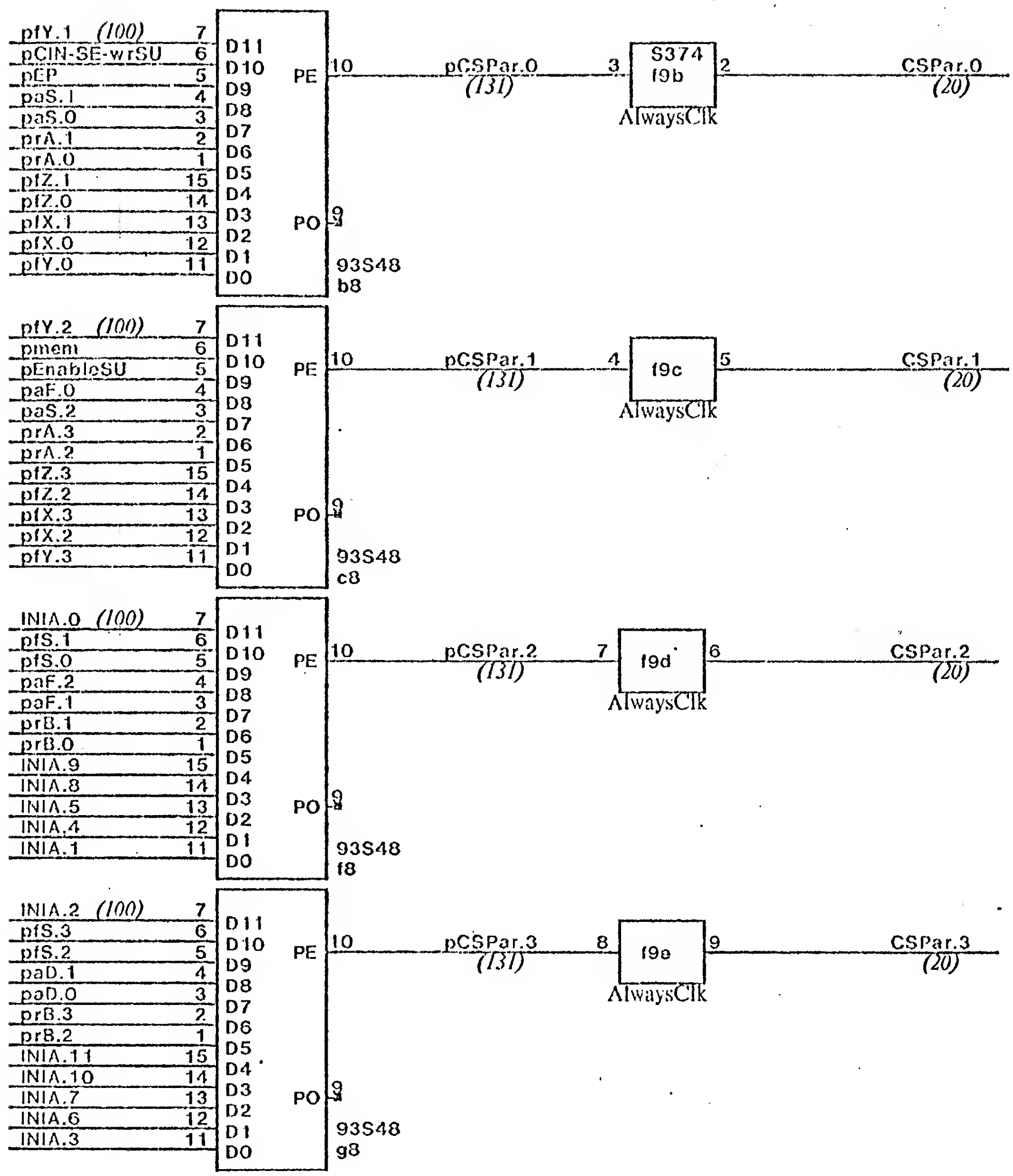

NIA.0d' 12 A11
NIA.1d' 13 A10
NIA.2d' 14 A9
NIA.3d' 15 A8
NIA.4d' 16 A7
NIA.5d' 17 A6
NIA.6d' 6 A5
NIA.7d' 5 A4
NIA.8d' 4 A3
NIA.9d' 3 A2
NIA.10d' 2 A1
NIA.11d' 1 A0
CSIn.0 11 DIN
CSWE.d' 8 WE'
GND 10 CS'
DO 7 pfY.0
i2147 a4
CSIn.1 11 DIN
DO 7 pfY.1
i2147 b4
CSIn.2 11 DIN
DO 7 pfY.2
i2147 c4
CSIn.3 11 DIN
DO 7 pfY.3
i2147 d4
CSIn.4 11 DIN
DO 7 INIA.0
i2147 e4
CSIn.5 11 DIN
DO 7 INIA.1
i2147 f4
CSIn.6 11 DIN
DO 7 INIA.2
i2147 g4
CSIn.7 11 DIN
DO 7 INIA.3
i2147 h4



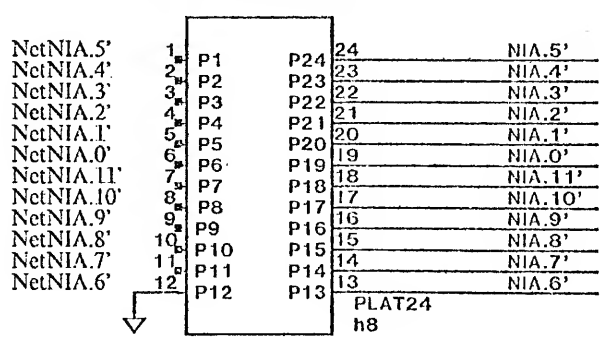
XEROX SDD	Project Dandelion	Control Store E [32-39]	File LionHead21.sil	Designer Garner	Rev B	Date 10/10/79	Page 21



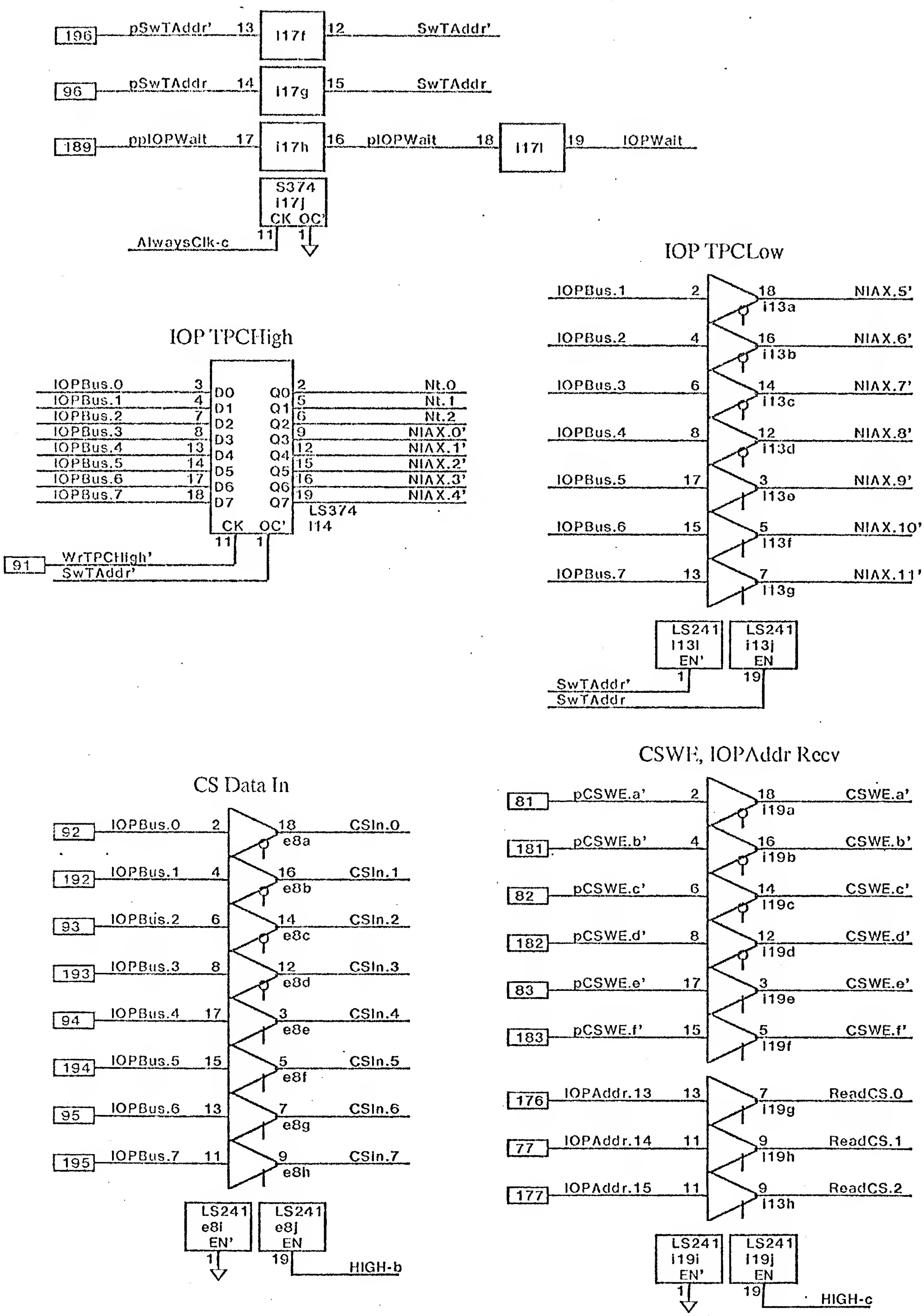
XEROX SDD	Project Dandelion	Control Store F [40-47]	File LionHead22.sil	Designer Garner	Rev B	Date 10/10/79	Page 22



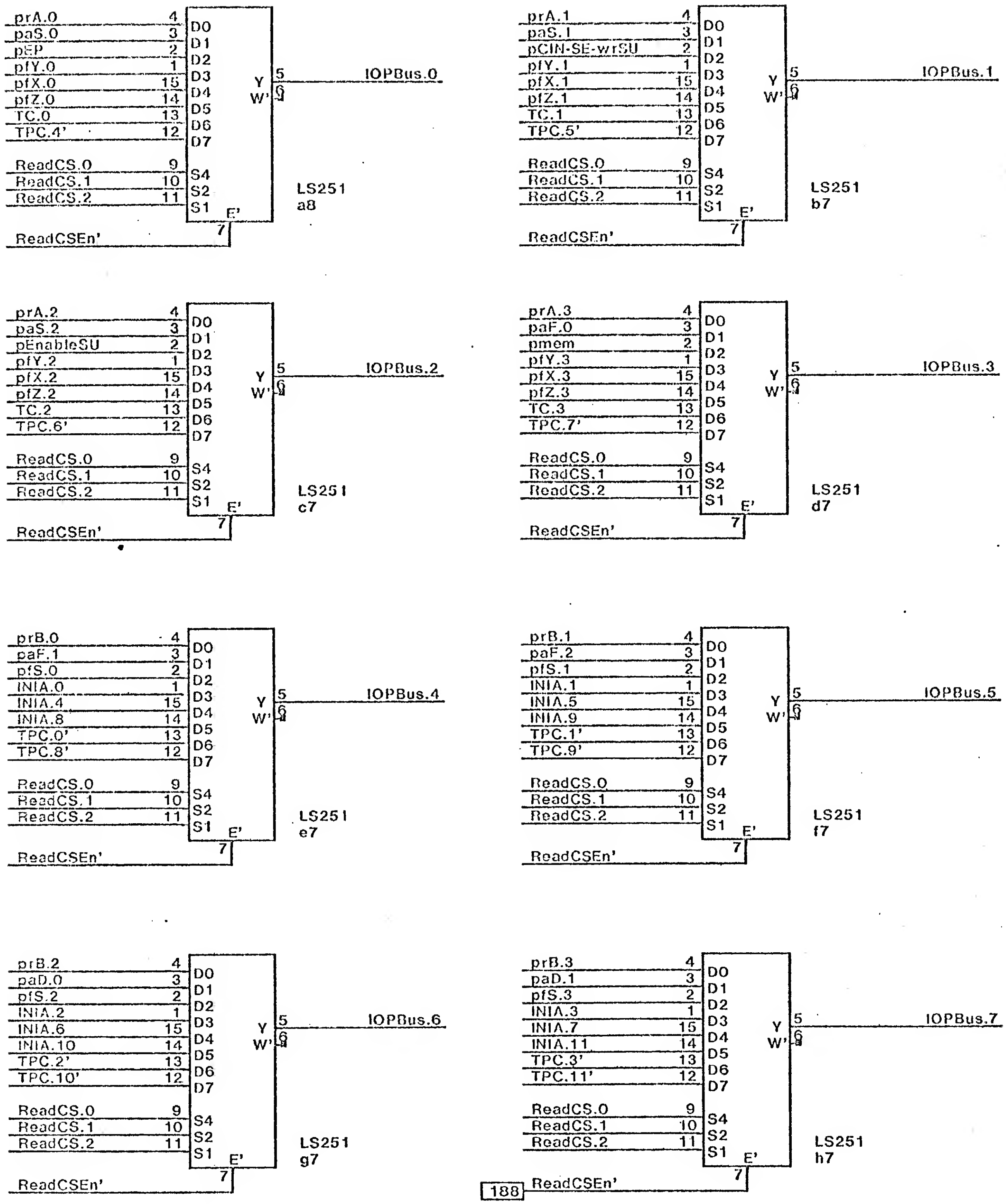
CS NIA Line Matching



See NetNIA.sil for defintion of NetNIA nets.
P12 is shown grounded so the trace will not be cut.

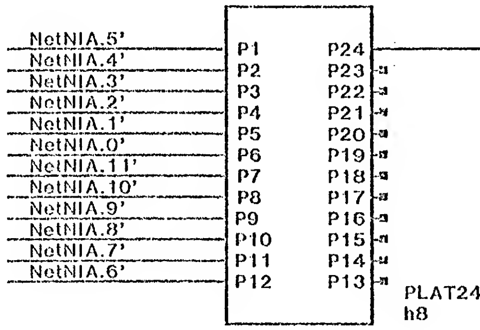


XEROX SDD	Project Dandelion	Tasks, IOP TPC-TC Control	File LionHead24.sil	Designer Garner	Rev B	Date 10/10/79	Page 24

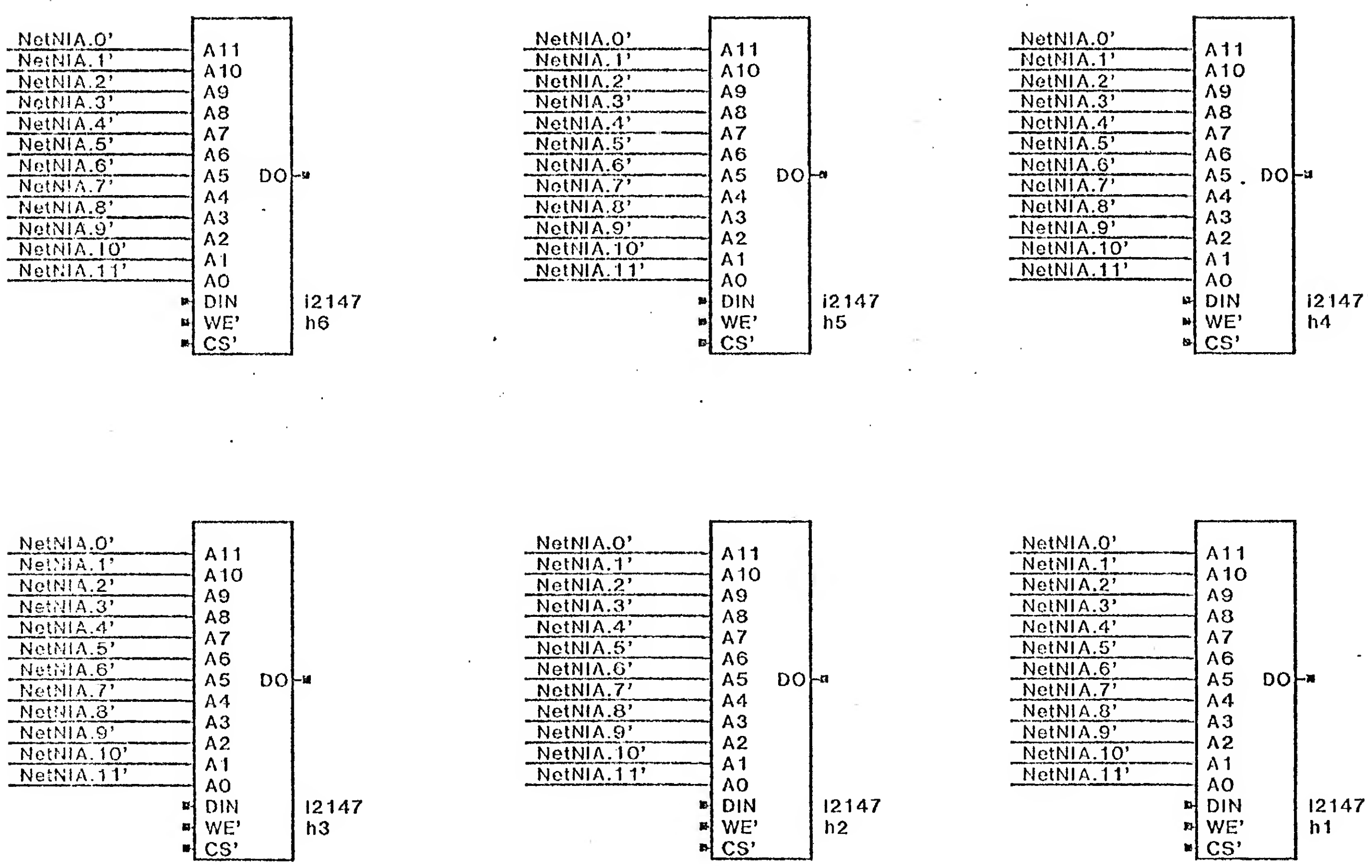


XEROX SDD	Project Dandelion	CS Read	File LionHead25.sil	Designer Garner	Rev B	Date 10/10/79	Page 25

This diagram defines the NetNIA.wl wirelist. This list should be wired before LionHead.wl.
Except for the first entry in each net, each node should be welded off center and towards the closest edge of the board



Just cut the ground connection (which is really a NetNIA line), the LionHead wire list will cut the VCC connection.
(The LionHead wire list should not try to cut the GND again, since it will have been connected to NetNIA.11')



XEROX SDD	Project Dandelion	NetNIA circuits	File LionHead26.sily	Designer Garner	Rev A	Date 8/11/79	Page 26

Rev A to Rev B (9 Oct 79)

1. Added timing info to all pages. Divided page 14 into 14 and 15, renumbering original 15-25.

Page 2: a. 1K pullup pack changed to 22-330 resistor pullup/pulldown. Ground to P8.
b. S09 changed to S03. Bits into Q ends inverted now. CIN-SE-wrSU' and pc16' necessary for CIN-SE.

Page 4: a. stackP read (instead of NstackP) onto X-bus (allows stackP in arithmetic operations).
b. RH[0-3] moved to d17.

Page 5: a. IBProm changed: SelIB0' and SelIB1' are now used to immediately select either IB[0] or IB[1].
IB←' input removed and replaced with EKErrc2 (to cancel GoodIBDispc2, instead of at the pNIA S64's.
Interchanged IBPtr.0 and IBPtr.1, deleted IBPtr.1'.

Page 6: a. Changed pin4 of f19 from Y.4 to Y.3
b. Interchanged fZ.3 and fZ.2 on 25S10's

Page 7: a. Changed ErrInt Status to ErrIBPtr, i.e. subsituted IBPtr.0 for MesaInt' and IBPtr.1 for CSParErr'.

Page 8: a. Changed mem from pin 14 to pin 113.
b. Moved CIN-SE-wrSU from c9 to b9, creating CIN-SE-wrSU'. paF.0 took CIN's place. aD.0 was moved to aD.1, and aD.1 to aF.0
c. Changed MAR← to MAR←', discontected MAR← from backplane.

Page 9: a. On b11(fZNorm), changed AlwaysNI' to IBPtr←0'; fS.2 to fS.2'; and moved all outputs up one position.

Page 10: a. Added S04 inverter for aD.0', moved RH← to page 16.
b. Changed S20's to S08 + S10's, opening up an S10 for use.

Page 11: a. Replaced Cycle1 test with CSParErr and NibCarry test with MesaInt.

Page 12: a. at pNIA[0-3] changed pin 6 from GND to HIGH (to distinquish no interrupt, empty buffer from error trap at 0)
b. Rearranged pNIA[8-11] S64 inputs: EKErrc2 should have zeroed the dispatch/branch bits also.

Page 13: a. Moved Link.3' connection to pullup pack since it is now 220-330 Pullup-down.
b. Changed NIA's SB inputs from Swc3 to Swc2.

Page 14: a. Enlarged Schedule Prom, adding RefReq' input. Pullup connections to requests from Options board.
b. Changed all inputs to SwitchProm (see programs).

Page 15: a. MemCSErrProm renamed CSIntProm since MemErr moved out to S08 and MesaInt moved in.
b. StackVirtErrProm renamed StackVirtProm, ClrIntErr' input not needed.
c. ErrorProm inputs changed: Nt = Emu to Ct = Emu.
d. KEProm renamed KernPC16Prom since MesaInt moved out. KernReq' an input now.

Page 16: a. WriteIB qualifier changed from S08 to S260 with ppCLK--reduced IB's large hold time.
b. WrTPCLow inverted, RH← moved here.
c. Detection of Low bank changed to S260 (freeing up and S02 and S08).

Page 25: a. LS251 inputs rearranged so read data is identical to write data format.

1 101	10	20	30	40	50 150	51 151	60	70	80	90	100 200

	a	b	c	d	e	f	g	h	i	
	S02 @ pCLK(2),Pt,	S257 0, Byte	25S10 LRotn.0	25S10 LRotn.1	25S10 LRotn.2	25S10 LRotn.3	LS377 Tasks	F93427 SchedProm	LS241 CS-IOP Recv	19
18	S138 IOOut	S138 IOIn	S241 Zero on X	S241 RH on Xhigh	S241 Nibble, Pt	S257 ibLow, ibHigh	HM7649 IBProm	F93453 ErrorProm	F93427 SwProm	18
17	S138 IOOut	S138 IOIn	F93453 StackVirtErr	AM29701 RH[0-3]	S240 ErrInt, stackP	S257 0, ibHigh	S373 IB[1]	S374 WaitClks	S374 @ AlwaysClks	17
16		LS283 NstackP	25S09 stackP	S260 ProcLow, WrIB	AM29701 RH[4-7]	LS377 IBFront	S373 IB[0]		F93453 Kpc16Prom	16
15	S00 Map,Nib', pMAR',X←C	25S09 SUAddrHigh	F93422 SU[0-3]	F93422 SU[4-7]	F93422 SU[8-11]	F93422 SU[12-15]	S374 AlwaysClks	F93427 CSIntProm		15
14	S138 fYNorm(Req)	25S09 SUAddrLow	S04 @ Mar←,RH←,c3, wrTPC,aD.0,	S00 WrSU,WrLink, WrTC,WrRH	k5 IDM2901A-1 [12-15]		S253 R Ends	S03 Q Ends, CIN-SE	LS374 IOPTPCHigh	14
13	S138 fYNorm	S08 paShO,Byte, fZHigh,MemErr	S51 WriteTPC', Waitc1'	S04 AlwysClk(3), WaitClk,C1,c2			S182 LookAhead	220-330 Resistors	LS241 IOPTPCLow	13
12	a S138 fX	b m12 IDM2901A-1 [0-3]	c		e n12 IDM2901A-1 [4-7]	f	g p12 IDM2901A-1 [8-11]	h		12
11	S10 XBus←IB, LRotn,XByte'	S138 fZNorm	S32 EnDispBr	S10 @ pTC.3,Wait,	S151 DispBr.3A	S151 DispBr.2	S64 pNIA.8	S64 pNIA.9	S64 pNIA.10	11
	S00 sh,Cin←16, pop,push	25S09 aSh,aFh	S374 rB,aSl,aSh	S00 pTC.0,1,2, Carry	S151 DispBr.3B	S258 DispBr.01	LS158 pNIA[4-7]	LS158 pNIA[0-3]	S64 pNIA.11	10
9	S374 fX, fZ	S175 Cin',fY.0,fS	S374 Misc,fY,fS	S374 rA,aD,aFl	AM29700 Link	S374 TC, CSPar	AM29701 TC	S374 NIAX[8-11]	25S09 NIA[8-11]	9
8	a LS251 IOPBus.0	b 93S48 pCSPar.0	c 93S48 pCSPar.1		e LS241 CSIn	f 93S48 pCSPar.2	g 93S48 pCSPar.3	h 18 ohm resistors		8
7	i2147L pfZ.0	LS251 IOPBus.1	LS251 IOPBus.2	LS251 IOPBus.3	LS251 IOPBus.4	LS251 IOPBus.5	LS251 IOPBus.6	LS251 IOPBus.7	25S09 NIA[0-3]	7
6	i2147L pfX.0	i2147L pfZ.1	i2147L pfZ.2	i2147L pfZ.3	i2147L INIA.8	i2147L INIA.9	i2147L INIA.10	i2147L INIA.11	25S09 NIA[4-7]	6
5		i2147L pfX.1	i2147L pfX.2	i2147L pfX.3	i2147L INIA.4	i2147L INIA.5	i2147L INIA.6	i2147L INIA.7		5
4	i2147L pfY.0	i2147L pfY.1	i2147L pfY.2	i2147L pfY.3	i2147L INIA.0	i2147L INIA.1	i2147L INIA.2	i2147L INIA.3	S374 NIAX[0-7]	4
3	i2147L pEP	i2147L pCIN-SE-WrSU	i2147L pEnSU	i2147L pmem	i2147L pfS.0	i2147L pfS.1	i2147L pfS.2	i2147L pfS.3	AM29701 TPC[8-11]	3
2	i2147L paS.0	i2147L paS.1	i2147L paS.2	i2147L paF.0	i2147L paF.1	i2147L paF.2	i2147L paD.0	i2147L paD.1	AM29701 TPC[0-3]	2
1	i2147L prA.0	i2147L prA.1	i2147L prA.2	i2147L prA.3	i2147L prB.0	i2147L prB.1	i2147L prB.2	i2147L prB.3	AM29701 TPC[4-7]	1
	a	b	c	d	e	f	g	h	i	

DIP Orient.

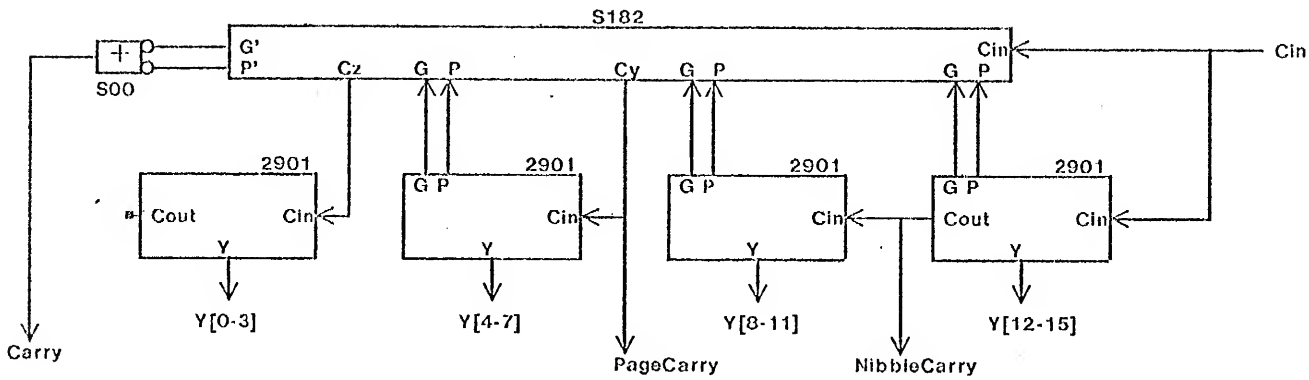
I/O Connector Area (Top) | I/O Connector Area (Bottom)

XEROX SDD	Project Dandelion	Chip & Function Layout	File LionHead28.sily	Designer Garner	Rev C	Date 10/30/79	Page 28

YH,,Y Bus MAR← timing:
For high-half ALU, operation is 0 or B.

6 7	0 7	8 11	12 15
17[3] Clk -> rB 25 rB -> YH 5 YH bus 47[3] = 50 nS	72 nS (See Ybus logic timing)	59 disable X 67 D -> Y 126 nS	59 disable X (See X bus timing) 42 D -> Y (See Y bus arithmetictiming) 101 nS

Y bus Arith timing:
Ripple carry is used in low half of ALU, and lookahead in high half.



X bus arithmetic: Ybus ← XBus + A

x	Xbus time
30	D -> G,P[4-11]
7[2]	G,P -> Cin.7, Cin.3
25	Cin -> Y[0-7]
10	Y bus
72[2]	= (74 + x) nS

x	Xbus time
32	D -> Cout[12-15]
25	Cin -> Y[8-11]
10	Y bus
(67 + x) nS	

x	Xbus time
32	D -> Y[12-15]
10	Y bus
(42 + x) nS	

Register Arithmetic: Ybus ← A + B

max(109, 95)

17[3]	↑ -> rB
45	rB -> G,P
7[2]	G,P -> Cin.7, Cin.3
25	Cin -> Y[0-7]
10	Y bus
104[5]	= 109 nS

48	↑ -> CIN-SE (see p. 2)
11[1]	S182 Cin -> Cin.7, Cin.3
25	Cin -> Y[0-7]
10	Y bus
94[1]	= 95 nS

max(105, 99)

17[3]	↑ -> rB
50	rB -> Cout[12-15]
25	Cin -> Y[8-11]
10	Y bus
102[3]	= 105 nS

48	↑ -> CIN-SE (see p. 2)
16	Cin[12-15] -> Cout
25	Cin -> Y[8-11]
10	Y bus
99	nS

max(83, 80)

17[3]	↑ -> rB
50	rB -> Y[12-15]
10	Y bus
77[3]	= 80 nS

48	↑ -> CIN-SE (see p. 2)
25	Cin -> Y[12-15]
10	Y bus
83	nS

Ybus Logic Timing:

Xbus Logic Ybus ← Xbus .or. 0	x 32 10 (42 + x) nS	X bus D -> Y Ybus
Register Logic: Ybus ← A .or. B	17[3] 50 10 77[3] = 80 nS	↑ -> rA, rB rB -> Y Ybus
A pass around (aD=2): Ybus ← A	17[3] 40 10 67[3] = 69 nS	↑ -> rA rA -> Y (via A bypass) Ybus

X bus Source timing:

Xbus ← A LRotn {A bypass}	69 ↑ -> Y (see above) 12 25S18 prop (= 22 nS) 10 Xbus 91 nS		
Xbus ← (A or B) LRotn	80 + 22 = 102 nS	(For more LRotn timing, see p. 6)	
Xbus ← (A + B) LRotn	109 + 22 = 131 nS	105 + 22 = 127 nS	89 + 22 = 111 nS
Xbus ← SU	75 nS (see p. 3)		
Xbus ← RH[B]	64 nS (see p. 4)	Xbus[0-7] ← 0 = 55 nS (see p. 6)	
Xbus ← IB	64 nS (see p. 5)	Xbus[0-7] ← 0 = 55 nS (see p. 6)	
Xbus ← MD	97 nS max		
Xbus ← 4-bit constant	50 nS (see p. 7)	Xbus[0-7] ← 0 = 55 nS (see p. 6)	
Xbus ← 8-bit constant	56 nS (see p. 7)	Xbus[0-7] ← 0 = 55 nS (see p. 6)	
Xbus ← ErrIBStkP	59 nS (see p. 4)	Xbus[0-7] ← 0 = 55 nS (see p. 6)	
Xbus ← IOIn	34 ↑ -> ← IOIn' 4 backplane 15 S240 EN' to X 10 Xbus 63 nS	34 ↑ -> ← IOIn' (see p. 9) 4 backplane 18 S374 EN' to X 10 Xbus 66 nS	

External Register Write Setups:

SU Write Setup (SU ← Ybus):	5[1] 39 44[1] = 45 nS	F93422 data setup (from beginning of write pulse) WE pulse width
RH Write Setup (RH ← Xbus):	20[1] = 21 nS	Am29700 data setup (from end of write pulse)
IB Write Setup (IB ← Xbus):	36 nS (see p. 5)	
IOOut Write Setup (IOOut ← Xbus):	equals setup time of receiving reg. data setup for LS374/LS273 = 20[2] = 22 nS	

Dispatch/Branch Condition Bits Setup:

7[1] S151/S258 in -> DispBr
5 DispBr -> pTC
6[1] pTC -> pNIA
5[1] 25S09/S374 setup
23[3] = 26 nS

D-input Setup Times:

Logic B ← Xbus .or. 0 B ← Xbus .xor. A	40 nS
Logic & Branch B ← Xbus .xor. A, ZeroBr	32 D -> F.0, F = 0 26 DispBr setup 58 nS
Logic & YDisp B ← Xbus .xor. A, YDisp	32 D -> Y 10 Ybus 26 DispBr setup 68 nS
Logic & Shifting B ← Xbus .or. A, LShift1	35 D -> R.3 15 RAM3 setup 50 nS
Logic & Rotating B ← Xbus .or. A, LRot1	35 D -> R.15 9[1] S253 in to R.0 15 RAM0 setup 59[1] = 60 nS

	Xbus[0-7]	Xbus[8-11]	Xbus[12-15]
Register Arithmetic B ← Xbus + A	30 D -> G,P 7[2] G,P -> Cin.3, Cin.7 35 Cin setup 72[4] = 74 nS	30 D -> Cout[12-15] 35 Cin.11 setup 65 nS	40 nS (Logic setup)
Register Arithmetic & ZeroBr B ← Xbus + A, ZeroBr	30 D -> G,P 7[2] G,P -> Cin.3, Cin.7 30 Cin -> F = 0 26 DispBr setup 93[2] = 95 nS max(95, 86, 58)	30 D -> Cout[12-15] 30 Cin -> F = 0 26 DispBr setup 86 nS	58 nS (Logic&Branch)
Register Arithmetic & NegBr B ← Xbus + A, NegBr	22 Cin -> F.0 = 95-8 = 87 nS		
Register Arithmetic & OvBr B ← Xbus + A, OvBr	25 Cin -> Ovr = 95-5 = 90 nS		
Register Arith & CarryBr, PgCarryBr B ← Xbus + A, CarryBr	30 D -> G,P 11[2] G,P -> G',P' 5 S00 in ->Carry 26 DispBr setup 72[2] = 74 nS (CarryBr)	30 D -> G,P 7[2] G,P -> PgCarry 26 DispBr setup 63[2] = 65 nS (PgCarryBr)	
Arithmetic & YDisp B ← Xbus + A, YDisp	Timing for X[0-7] does not affect YDisp		68 nS (Logic&YDisp)
Arithmetic & Shifting B ← Xbus + A, RShift1	30 D -> G,P 7[2] G,P -> Cin.3, Cin.7 35 Cin -> R.3 15 RAM3 setup 87[2] = 89 nS	30 D -> Cout[12-15] 35 Cin -> R.3 15 RAM3 setup 80 nS	50 nS (Logic&Shifting)
Arithmetic & Rotating B ← Xbus + A, RRot1	89 + 10 = 99 nS	80 + 10 = 90 nS	60 nS (Logic&Rotating)

R Register Cycle Times (Times for branching use ZeroBr, slowest of the branches)

Register Logic
B ← A .and. B

17[3] ↑ -> rA
60 rA setup
77[3] = 79 nS

Register Logic & Branch
B ← A .xor. B, ZeroBr,

17[3] ↑ -> rA
55 rA -> F = 0
26 DispBr setup
98[3] = 101 nS

Register Logic & YDisp
B ← A .xor. B, YDisp,

78 Ybus ← A .xor. B (see p.30)
26 DispBr setup
104 nS

A Bypass & YDisp
[] ← A, YDisp

69 Ybus ← A (see p.30)
26 DispBr setup
95 nS

Register Logic & Shifting
B ← A .or. B, LShift1

17[3] ↑ -> rA
55 rA -> R.3
15 RAM3 setup
87[3] = 90 nS

Register Logic & Rotating
B ← A .or. B, LRot1

17[3] ↑ -> rA
55 rA -> R.3
9[1] S253 In to R.0
15 RAM0 setup
96[4] = 100 nS

	bits[0-7]	bits[8-11]	bits[12-15]
Register Arithmetic B ← A + B	17[3] ↑ -> rA 45 rA -> G,P 7[2] G,P -> Cin.3, Cin.7 35 Cin setup 104[5] = 109 nS	17[3] ↑ -> rA 50 rA -> Cout[12-15] 35 Cin.11 setup 102[3] = 105 nS	79 nS (Reg Logic)
Register Arithmetic & Branch B ← A + B, ZeroBr	17[3] ↑ -> rA 45 rA -> G,P 7[2] G,P -> Cin.3, Cin.7 30 Cin -> F = 0 26 DispBr setup 127[5] = 132 nS	17[3] ↑ -> rA 30 rA -> Cout[12-15] 30 Cin -> F = 0 26 DispBr setup 103[3] = 106 nS	101 nS (Logic&Branch)
Arithmetic & YDisp B ← A + B, YDisp	Timing for X[0-7] does not affect YDisp		104 nS (Logic&YDisp)
Arithmetic & Shifting B ← A + B, RShift1	17[3] ↑ -> rA 30 rA -> G,P 7[2] G,P -> Cin.3, Cin.7 35 Cin -> R.3 15 RAM3 setup 107[5] = 112 nS	17[3] ↑ -> rA 30 rA -> Cout[12-15] 35 Cin -> R.3 15 RAM3 setup 97[3] = 100 nS	90 nS (Logic&Shifting)
Arithmetic & Rotating B ← A + B, RRot1	112 + 10 = 122 nS	100 + 10 = 110 nS	100 nS (Logic&Rotating)

X Operation	D setup	X Source: SU	* RH	MD	* Nibble	* Byte	* IB	zero [0-7]	A LRotn	IOIn	* ErrIBStkP	(A .or. B) LRotn	(A+B) LRotn
X ←		75	64	97	50	56	64	55	91	63	59	102	"
max (59, X←)		75	64	97	59	59	64	59	91	63	59	102	131 127 111
B←X .or. A	40	115	104	137	99	99	104	99	131	103	99	—	—
B←X .or. A, ZeroBr	58	133	122	155	117	117	122	117	149	121	117	—	—
[]←X .or.A, YDisp	68	143	132	165	127	127	132	127	159	131	127	—	—
B←X .or. A, LShift1	50	125	114	147	109	109	114	109	—	113	109	—	—
B←X .or. A, LRot1	59	134	123	156	118	118	123	118	—	122	118	—	—
MAR←X .or. A	78	153	142	—	137	137	142	137	—	141	137	—	—
MDR←X .or. A	45	120	109	—	104	104	109	104	—	108	104	—	—
SU←X .or. A	87	—	151	184	146	146	151	146	—	150	146	—	—
IOYOut←X .or. A	64	139	128	161	123	123	128	123	—	127	123	—	—
IOYOut←X .or. A	48	123	112	145	107	107	112	107	—	111	107	—	—
B←X + A	74 65 40	149 140 115	133 129 104	171 162 137	133 124 99	133 124 99	133 129 104	133 - -	165 156 131	137 128 103	133 124 99	—	—
B←X + A, ZeroBr	95	170	154	192	154	154	154	154	186	158	154	—	—
B←X + A, NegBr	87	162	151	184	146	146	151	146	178	150	146	—	—
B←X + A,PgCarryBr	65	140	129	162	124	124	129	—	156	128	124	—	—
B←X + A, CarryBr	74	149	133	171	133	133	133	133	165	137	133	—	—
B←X + A, YDisp	68	143	132	165	127	127	132	—	159	131	127	—	—
B←X + A, RShift1	89 80 50	164 155 125	148 144 114	186 177 147	148 139 109	148 139 109	148 144 114	148 - -	—	152 143 113	148 139 109	—	—
B←X + A, RRot1	99 90 60	174 165 135	158 154 124	196 187 157	158 149 119	158 149 119	158 154 124	158 - -	—	162 153 123	158 149 119	—	—
MAR←X + A	- 78 78	153	142	—	137	137	142	137	—	141	137	—	—
MDR←X + A	77 70 45	152 144 120	136 134 109	—	136 129 104	136 129 104	136 134 109	136 - -	—	135 133 108	136 129 104	—	—
SU←X + A	119 112 87	—	178 176 151	216 209 184	178 171 146	178 171 146	178 176 151	178 - -	—	182 174 150	178 171 146	—	—
IOYOut←X + A LS374	96 89 64	171 164 139	155 153 128	193 186 161	155 148 123	155 148 123	155 153 128	155 - -	—	159 152 127	155 148 123	—	—
IOYOut←X + A S374	80 73 48	155 148 123	139 137 112	177 170 145	139 132 107	139 132 107	139 137 112	139 - -	—	143 136 111	139 132 107	—	—
[] ← X, XDisp	26	101	90	123	85	85	90	85	117	89	85	128	157 153 137
RH ← X	21	96	—	118	80	80	85	80	112	84	80	123	152 148 132
IB ← X	36	111	100	133	95	95	100	95	127	99	95	138	167 163 147
IOXOut←X (LS374)	22	97	84	117	79	79	84	79	111	83	79	122	151 147 131
IOXOut←X (S374)	6	81	70	103	65	65	70	65	97	69	65	108	137 131 117

The 3 numbers for arithmetic operations correspond to bits[0-7], bits[8-11], & bits[12-15], respectively.

			Y Source		
		setup	A .or. B	A (bypass)	A + B
	Y ←		78	69	109 105 89
Y Operation	MAR ← *	36 11 36	114 89 114	105 80 105	114 116 125
	MDR←	3	81	72	112 108 92
	SU←	45	123	114	154 150 134
	[]← , YDisp	26	104	95	135 131 115
	IOYOut← (LS374)	22	100	91	131 127 111
	IOYOut← (S374)	6	84	75	115 112 95

* Bits[0-7] have timing of Y ← (B .or. 0), except in the A bypass case.

X bus loading

(for X[12-15] since these bits have the greatest loading)

Source	Sink	Part	Source Drive	Sink Load
	D-input	IDM2901A-1		.4/.18
	RH	Am29701		.2/.125
	IB	S373		1/.125
	IOPOData	LS374		.4/.2
	IOPCtl	LS273		.4/.2
	KOData	S374		1/.125
	KCtl	LS273		.4/.2
	XOData	S374		1/.125
	XCtl	LS273		.4/.2
	POData	LS374		.4/.2
	PCtl	LS273		.4/.2
SU		93422	104/8	1/.025
LRotn		Am25S10	130/10	1/.025
ErrIBStkp		S240	60/32	1/.025
RH		S241	60/32	1/.025
IB		S257	130/10	1/.025
Nibble		S241	60/32	1/.025
MD		S240	60/32	1/.025
IOPIData		S374	130/10	1/.025
IOPStatus		S240	60/32	1/.025
XIData		S374	130/10	1/.025
XStatus		S240	60/32	1/.025
KIData		S374	130/10	1/.025
KStatus		S240	60/32	1/.025
PStatus		S240	60/32	1/.025
MStatus		S240	60/32	1/.025
Min Source Drive		S240/93422	60/8	
Total Sink Load				21/2.25

Table Entries: High U.L./ Low U.L.

1 High U.L. = 50 uA
1 Low U.L. = 2.0 mA

Y bus Loading

Source	Sink	Part	Source Drive	Sink Load
Y-output		IDM2901A-1	32/10	
	LRotn	Am25S10		1.5/1.5
	SU	93442		.8/.15
	stackP	25S09		1/1
Y.4	MAR	S253		3(1/1)
Y.4	MDR	S373		1/.125
Y.4	MCtl	S138		1/1
	DOAData	S373		1/.125
	DOBData	S374		1/.125
	DCtl	LS273		.4/.2
Total Sink Load				10.7/7.3

Table Entries: High U.L./ Low U.L.

1 High U.L. = 50 uA
1 Low U.L. = 2.0 mA

Xerox Corporation
701 South Aviation Boulevard
El Segundo, California 90245

XEROX

MATERIAL LIST

ML	Drawing No.	Rev. C

Rev.

Dwg. No.

ML

Drawing Title

Dandelion CP Preliminary Parts List

Revision B

(for [Iris]<Workstation>LH>LionHead-C.dm)

(filed on >LH>CPPartsl-C.sil)

* indicates change from last rev.

Model No.	Date	Sheet
	30 Oct 79	1 of 2

Item No.	Drawing Title		Drawing No.	No. Req.	Remarks
	Integrated Circuit	IDM2901A-1		4	National or Am2901C
		AM29700		1	16x4 Non-Inv OC Ram
		AM29701		6	16x4 Non-Inv 3-S Ram
		AM93S48		4	12-input parity
		AM25S09	*	7	or 74S399
		AM25S10		4	4-bit shifter
		i2147L		48	low power 2147
		F93422		4	256x4 Ram
		F93427		3	256x4 Prom
		F93453		3	1024x4 Prom
		HM7649		1	512x8 Prom
		SN74S00		4	
		SN74S02		1	
		SN74S03		1	
		SN74S04		1	
		SN74S08		1	
		SN74S10		2	
		SN74S32	*	1	
		SN74S51		1	
		SN74S64	* (no more S74)	4	
		SN74S138		8	
		SN74S151		3	
		SN74S175		1	
		SN74S182		1	
		SN74S240		1	
		SN74S241		3	
		SN74S253		1	
		SN74S257		3	
		SN74S258	*	1	
	Integrated Circuit	SN74S260		1	

Xerox Corporation
701 South Aviation Boulevard
El Segundo, California 90245

XEROX

MATERIAL LIST

ML	Drawing No.	Rev. C

Rev.

Dwg. No.

ML

Drawing Title

Dandelion CP Preliminary Parts List

Revision B

(for [Iris]<Workstation>LID>LionHead-C.dm)
(filed on >LID>CPParts2-C.sil)

Model No.	Date	Sheet
	30 Oct 79	2 of 2

Item No.	Drawing Title		Drawing No.	No. Req.	Remarks
	Integrated Circuit	SN74S373		1	
		SN74S374	*	9	
		SN74LS158		2	
		SN74LS241		3	
		SN74LS251		8	
		SN74LS283	* no more LS352	1	
		SN74LS374		1	
	Integrated Circuit	SN74LS377	* no more LS399	2	
	Capacitor	.1 uF bypass, 25V			1 per x chip positions
	DIP Resistor Network	220-330 ohm pullup/pulldown 2% Allen Bradley 316E221331		1	
	DIP Resistor Network	8- 22 ohm resistors 2% Allen Bradley 316B220		2	Prefer lower value (~12 ohms)

Note: 74S189's can be used instead of Am29701's if S534's or 67S378's are better than S374's.

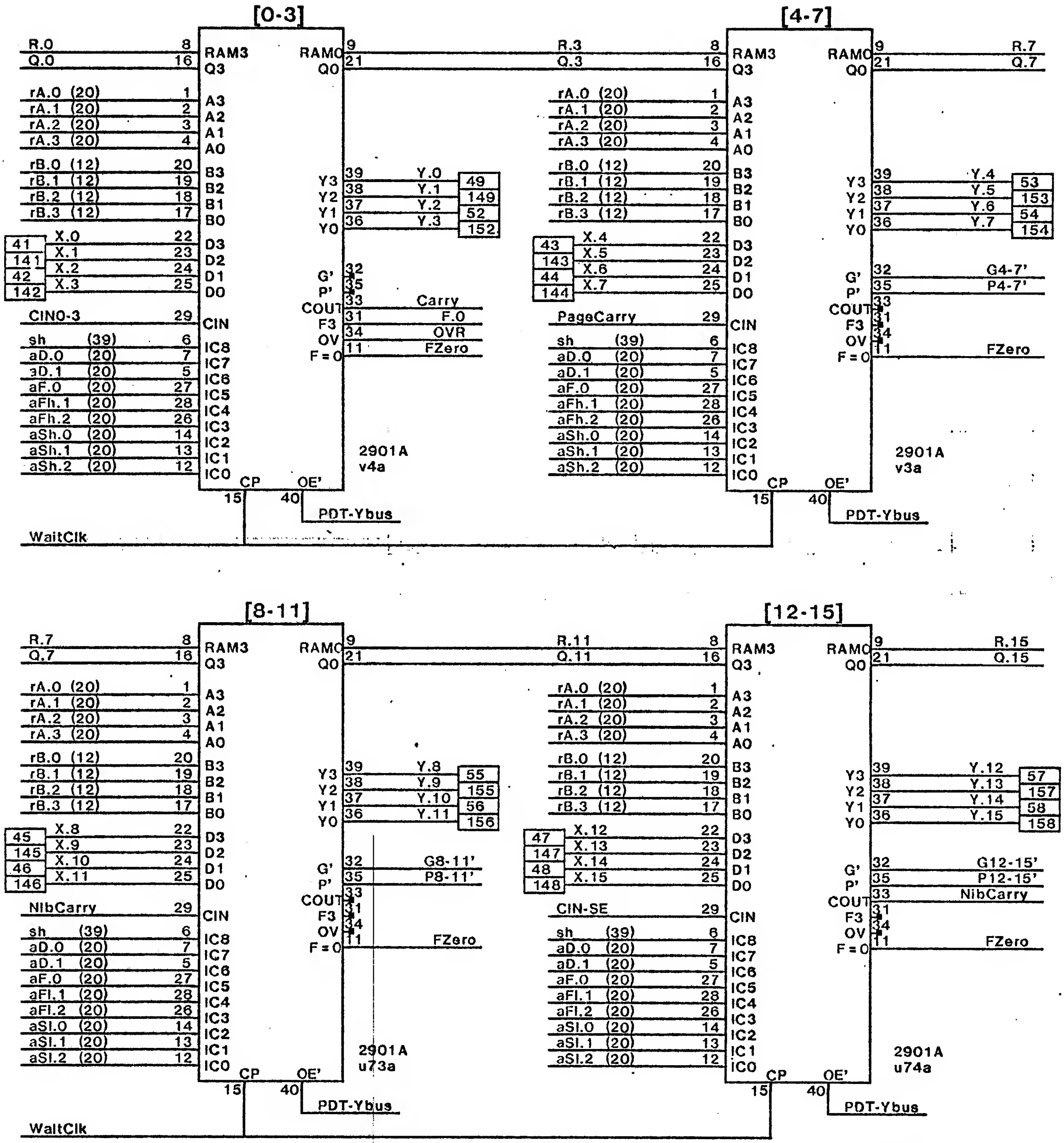
Similarily, S289's can be used instead of Am29700's if S534's or 67S378's (inv. oct. reg.) are better

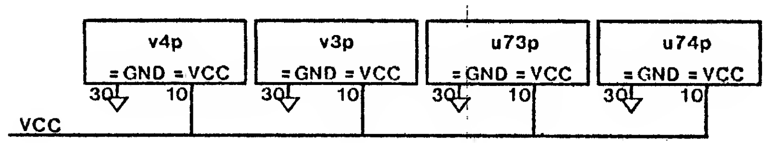
Dandelion Central Processor

pLionHead # #.sil are the printed circuit board schematics.
sLionHead # #.sil are the stichweld board schematics.
LionHead # #.sily are documentation pages.

Also see:

[Iris]<Workstation>LH>#LionHead-J.press — *s or p schematics*
[Iris]<Workstation>LH>CPProms-J.press — *Mesa prom programs*
[Iris]<Workstation>LH>DMR.press — *Dandelion Microcode Reference*
[Iris]<Workstation>LH>CPCheckOut.press
[Iris]<Workstation>LH>DLionIORules.press — *Rules for IO controllers*



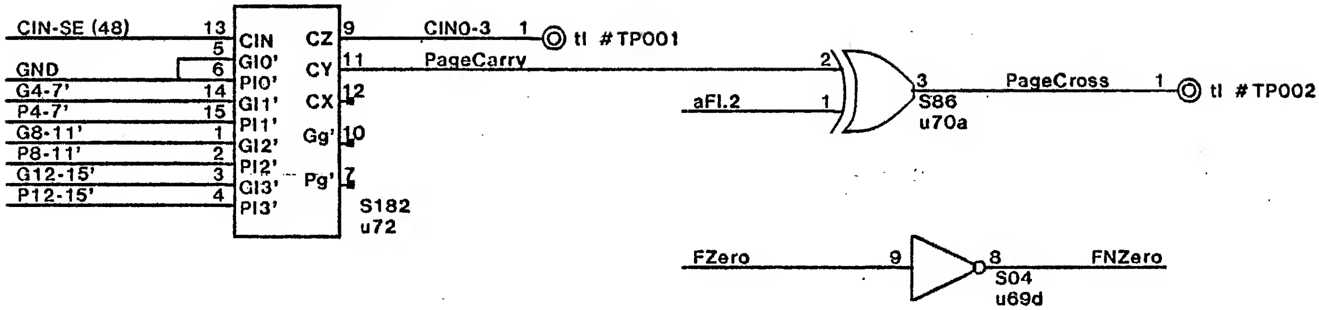


sh,,aD setup = 54 nS
aF setup = 45 nS
aS setup = 45 nS

rA	to G,P =	45	rA	to Y =	50 nS
D	to G,P =	30	D	to Y =	32
			Cn	to Y =	25
rA	to Cout =	50	aS	to Y =	40
D	to Cout =	32	aF	to Y =	35
Cin	to Cout =	16	aD	to Y =	25

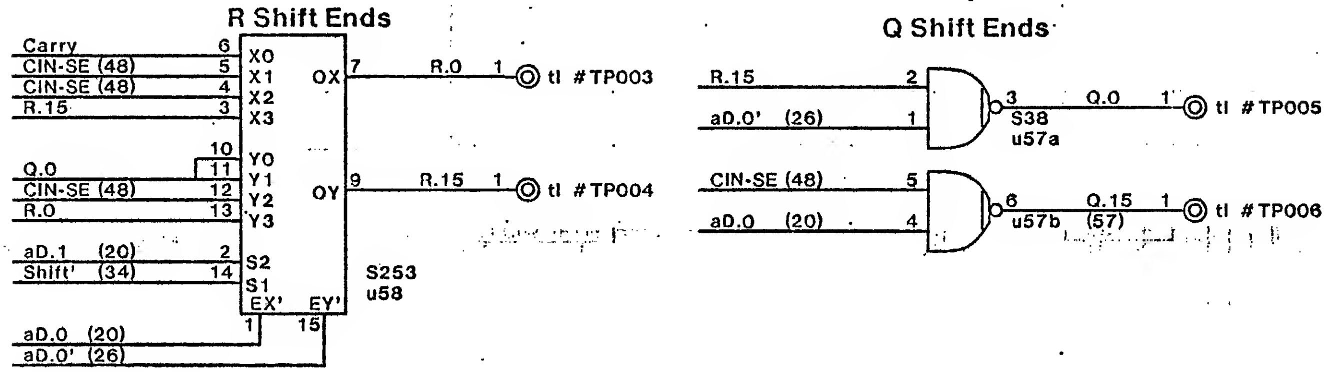
XEROX SDD	Project Dandelion	2901 chips	File pLionHead01.sil	Designer Garner	Rev J	Date 8/23/80	Page 01

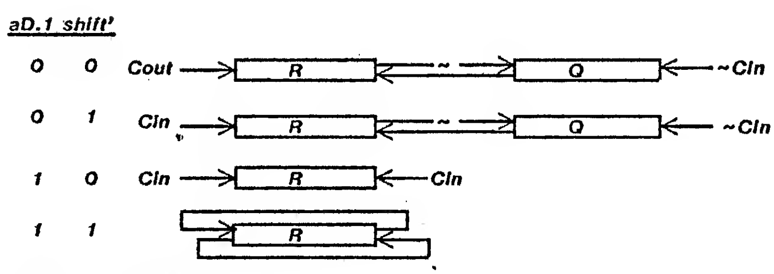
2901 Carry Lookahead



R Shift Ends

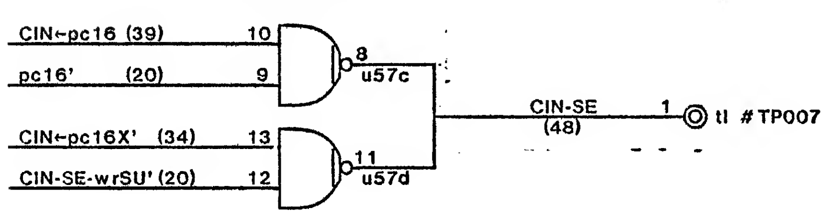
Q Shift Ends



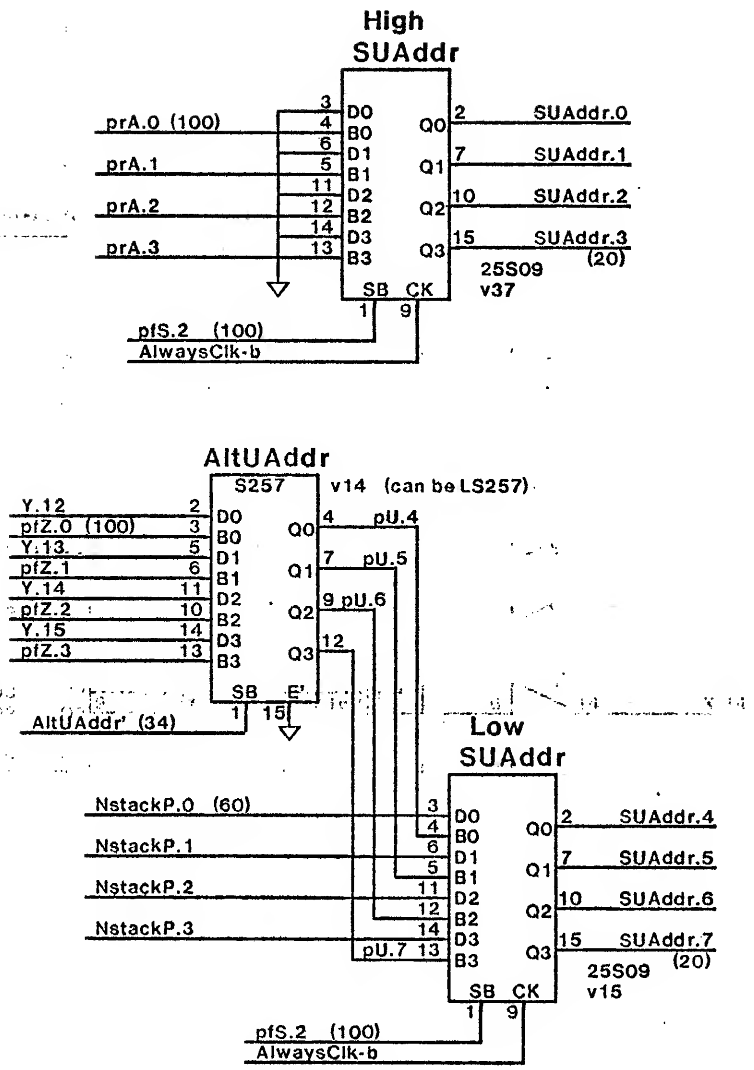


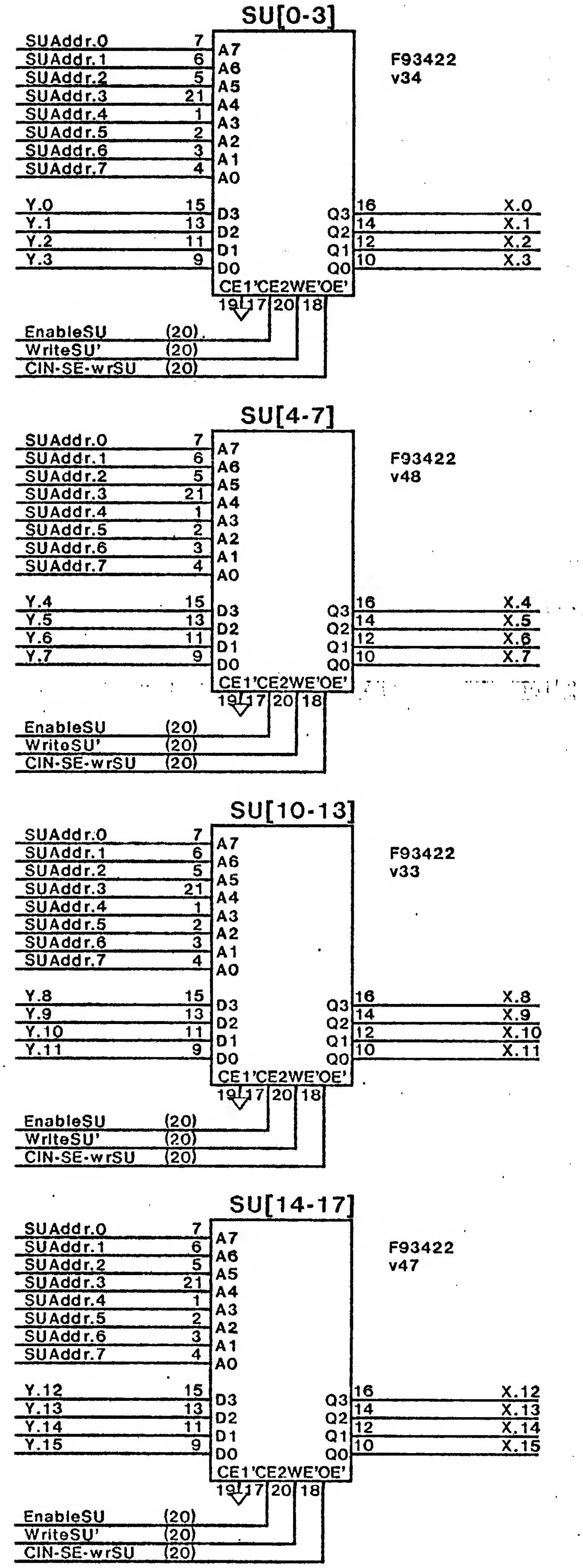
aD.0 = 0 implies right shift

Cin & Shift Ends



XEROX SDD	Project		File	Designer	Rev	Date	Page
	Dandelion	Lookahead, Shift Ends, Cin	pLionHead02.sil	Garner	J	8/23/80	02





SU X-bus disable

15[3] ↑ to CIN-SE-wrSU (tPLH)
30 Output Disable
10 X-bus
55[3] = 58 nS

XBus ← SU = max(75,60) nS

17[3] ↑ to SUAddr
45 tAA
10 X-bus
72[3] = 75 nS

17[3] ↑ to CIN-SE-wrSU/EnableSU
30 F93422 OE'/CE2 to X-bus
10 X-bus
57[3] = 60 nS

SU write setup

5[1] Data setup
39 WE
44[1] = 45 nS

AltUAddr setup

5[1] 25S09 setup
8[1] Y -> pU
13[2] = 15 nS (26 if LS257)

F93422 data t-hold = 5 nS

XEROX SDD	Project		File	Designer	Rev	Date	Page
	Dandelion	SU	pLionHead03.sil	Garner	J	8/23/80	03

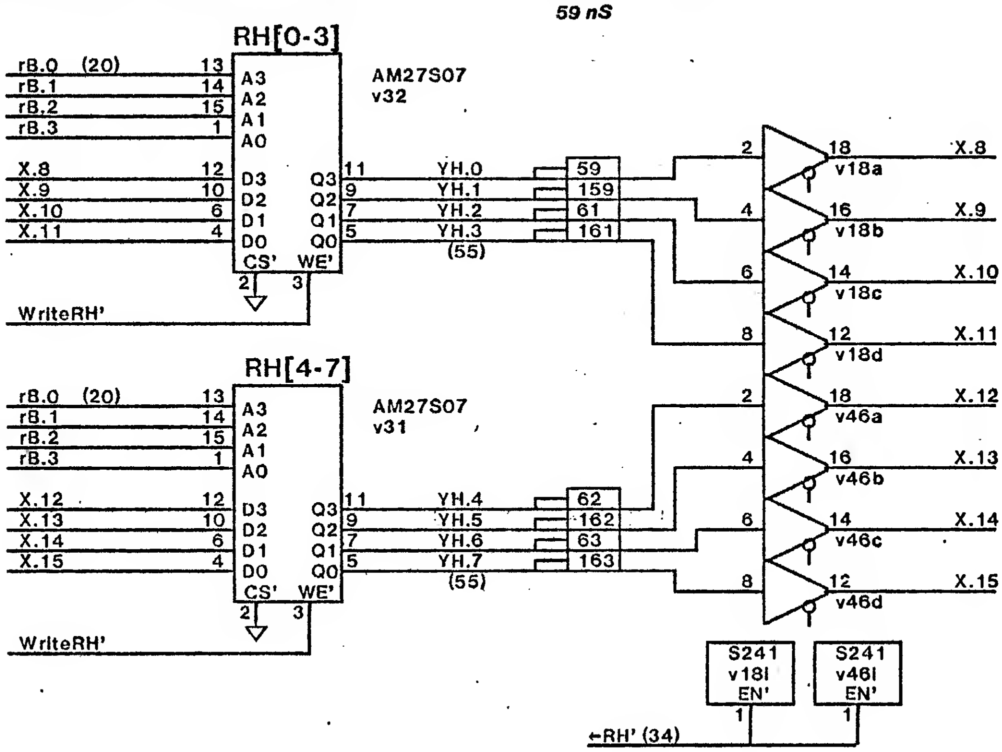
RH Setup

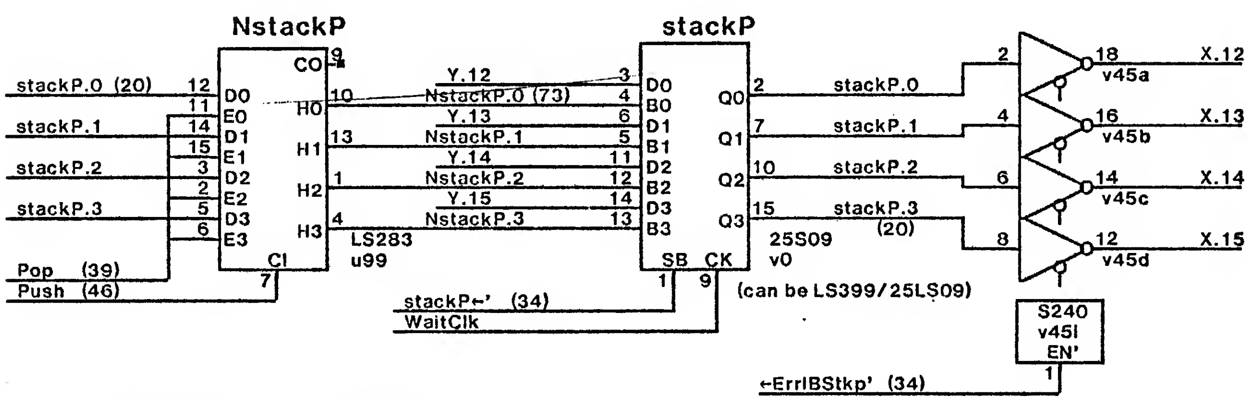
7[1]	S240
25[3]	S189 setup
32[4] = 36 nS	

XBus ← RH = max(64, 59) nS

17[3]	↑ to rB'
35	S189 tAA (wr recovery = 35 nS)
9	YH to X
10	X-bus
71[3] = 74 nS	
34	↑ to ←RH'
15	S241 EN' to X-bus
10	X-bus
59 nS	

If the S189 must be used intead of the 29701, then place S240 between Xbus & S189 inputs.





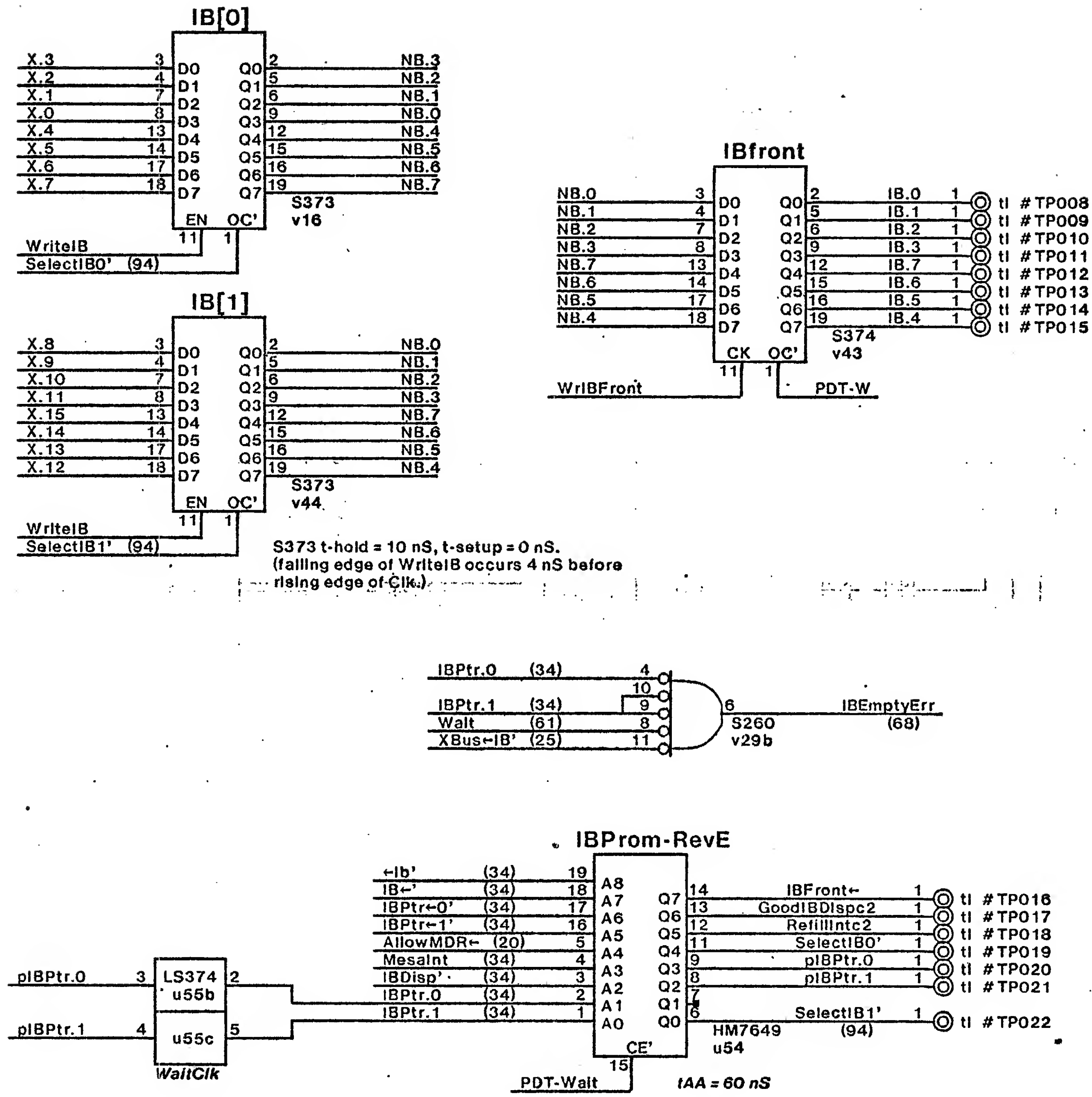
XBus ← stackP = max(59, 38) nS

17[3]	↑ to stackP
7	S240 data to X-bus
10	X-bus
34[3] = 38 nS	
34	↑ to ←ErrIntstackP'
15	S240 EN' to X-bus
10	X-bus
59 nS	

push timing

46	↑ to Push
24[3]	Push to NstackP
5[1]	25S09 setup
75[4] = 79 nS	

XEROX SDD	Project		File	Designer	Rev	Date	Page
	Dandelion	RH, stackP	pLionHead04.sil	Garner	J	8/24/80	04



IBFront ← Xbus = (x+37, x+36) nS

x	*Xbus to IB*	*x*	*Xbus to IB*	*94*	*WriteIB rises*
43	*WriteIB rises 43 nS before end of cycle*	*13[1]*	*S373 Data to NB*	*18[2]*	*S373 EN to NB*
- 6	*Difference between S373 "EN to Q" and "Data to Q" = 18[2] - 13[1] = 5 nS. Data can arrive 6 nS after WriteIB goes high.*	*20[2]*	*LS374 setup*	*20[2]*	*LS374 setup*
x+37 nS		*x+36 nS*		*132[4] = 136 nS*	

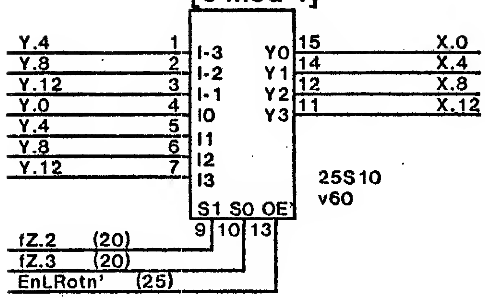
IBfront←IB[1]

34	*t to IBPtr←1'*
60	*tAA*
18[2]	*SelectIB1' to NB*
20[2]	*LS374 setup*
132[4] = 136 nS	

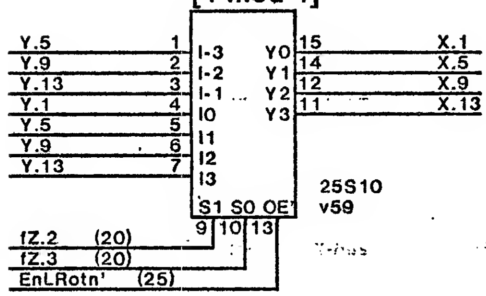
XEROX SDD	Project		File	Designer	Rev	Date	Page
	Dandelion	IB	pLionHead05.sil	Garner	J	8/23/80	05

LRotn

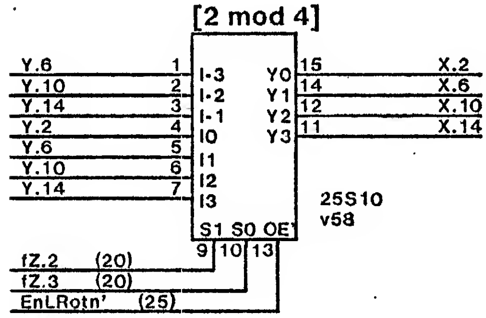
[0 mod 4]



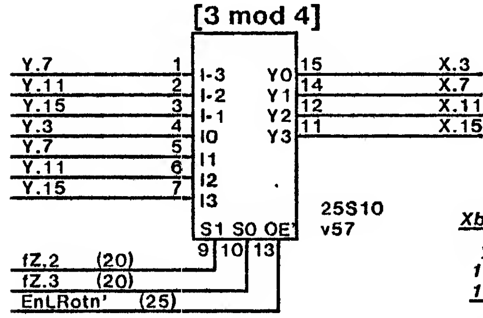
[1 mod 4]



[2 mod 4]



[3 mod 4]



fZ.2	fZ.3	
0	0	Left 0
0	1	Left 12
1	0	Left 8
1	1	Left 4

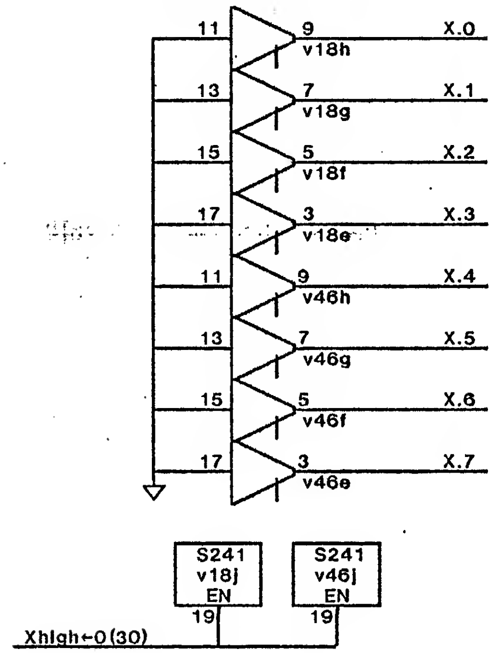
Zero disable X-bus

30	↑ to Xhigh←0
15	S241 EN to X-bus
10	X-bus
55 nS	

Xbus[0-7] ← 0

30	↑ to Xhigh←0
15	S241 OE
10	X-bus
55 nS	

Zero Hign XBus



Xbus ← Y LRotn = max (y + 22, 56, 50) nS

y	↑ to Y bus
12	25S10 data in to out
10	X-bus
y +	22 nS

25	↑ to EnLRotn'
21	25S10 OE
10	X-bus
56 nS	

20	↑ to fZ.2
20	25S10 Select to X-bus
10	X-bus
50 nS	

LRotn disable X-bus

25	↑ to EnLRotn'
15	25S10 OE' to X-bus
10	X-bus
50 nS	

XEROX SDD	Project		File	Designer	Rev	Date	Page
	Dandelion	X Bus: LRotn, RH, ZeroHighX	pLionHead06.sil	Garner	J	8/24/80	06

IB disable X-bus

25	↑ to XBus←IB'
14	S257 E' to X-bus
10	X-bus
49 nS	

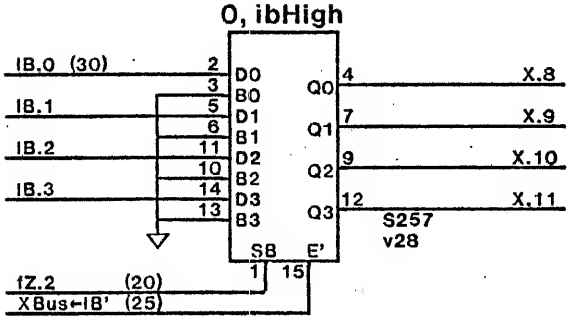
Byte disable X-bus

25	↑ to Nibble'
14	S257 E' to X-bus
10	X-bus
49 nS	

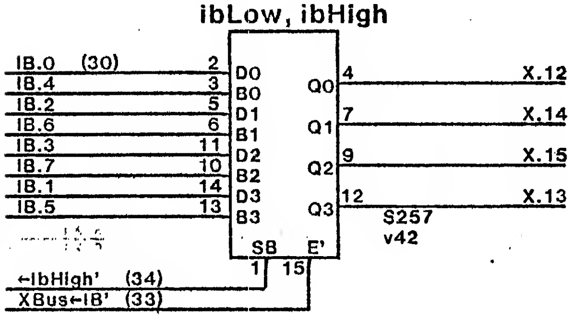
Nibble disable X-bus

25	↑ to Nibble'
15	S241 EN' to X-bus
10	X-bus
50 nS	

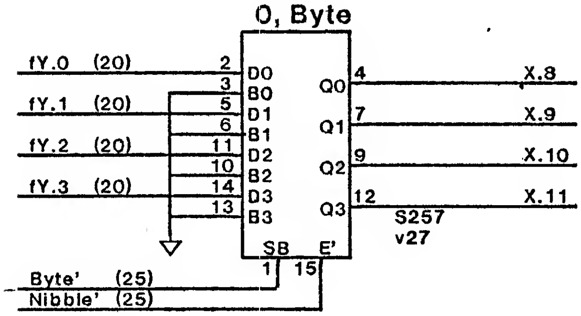
0, ibHigh



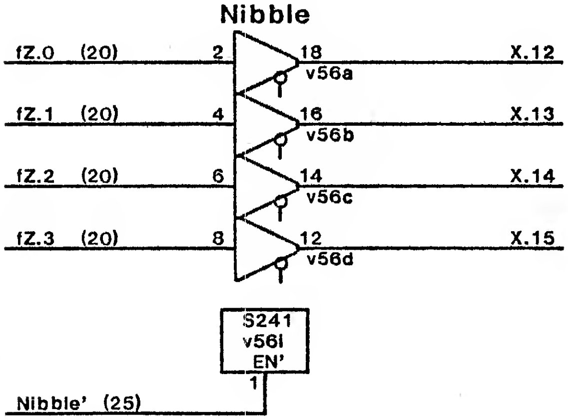
ibLow, ibHigh



0, Byte



Nibble



Xbus←IB = max(56,56,59) nS

34[4]	↑ to IB
8	S257 data to Xbus
10	X-bus
52[4] = 56 nS	
25	↑ to Xbus←IB'
21	S257 E' to Xbus
10	X-bus
56 nS	
34	↑ to←ibHigh'
15	S257 SB to Xbus
10	X-bus
59 nS	

Xbus ← Nibble = max(39, 50) nS

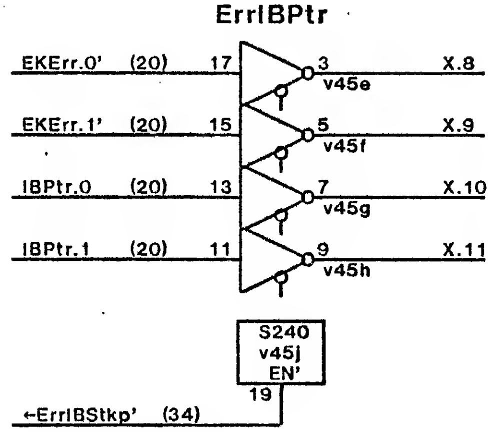
20	↑ to fZ
9	S241 data to X-bus
10	X-bus
39 nS	
25	↑ to Nibble'
15	S241 EN' to X-bus
10	X-bus
50 nS	

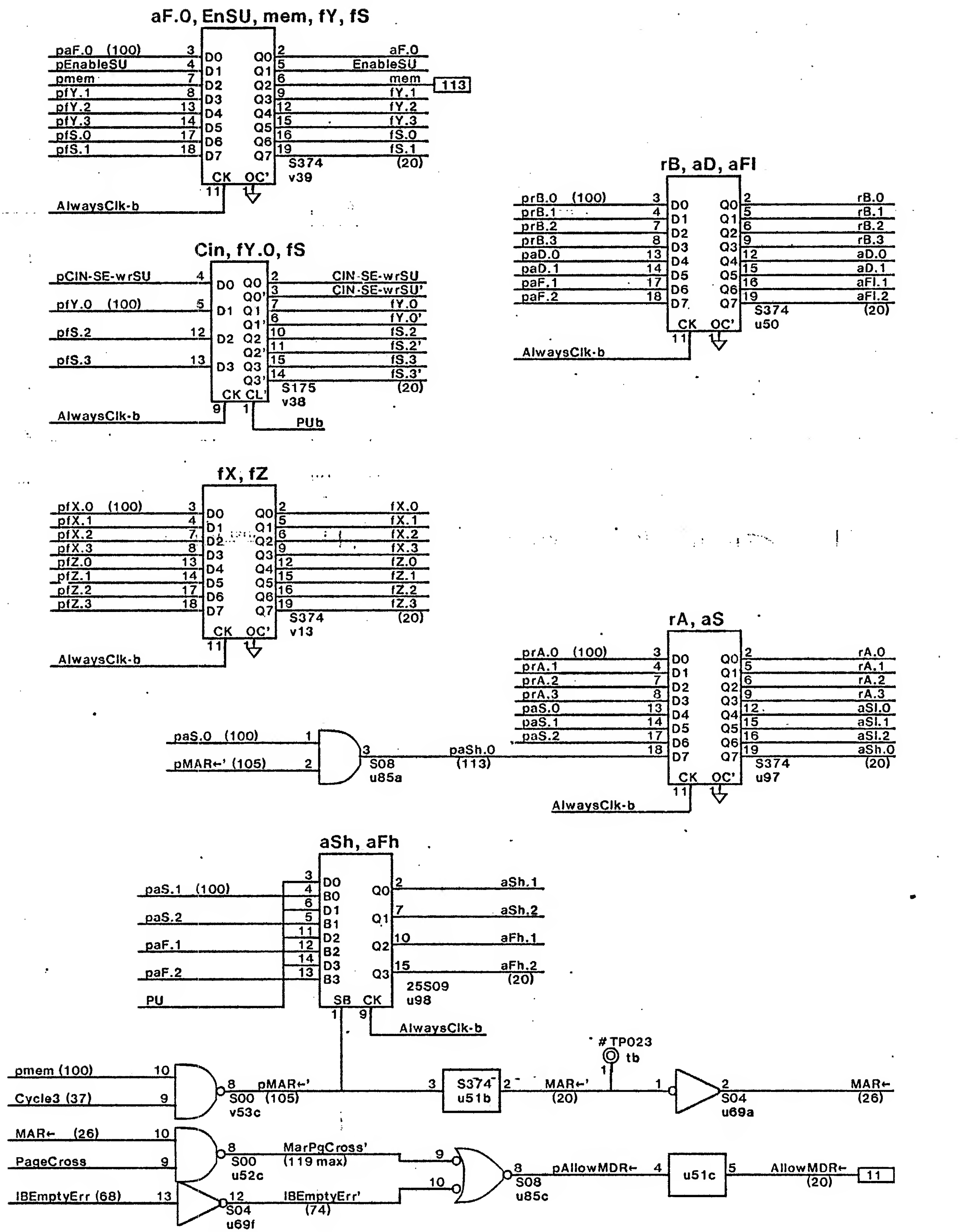
Xbus ← Byte = max(38, 56,50) nS

20	↑ to fY
8	S257 data to X-bus
10	X-bus
38 nS	
25	↑ to Nibble'
21	S257 E' to X-bus
10	X-bus
56 nS	
25	↑ to Byte'
15	S257 SB to Xbus
10	X-bus
50 nS	

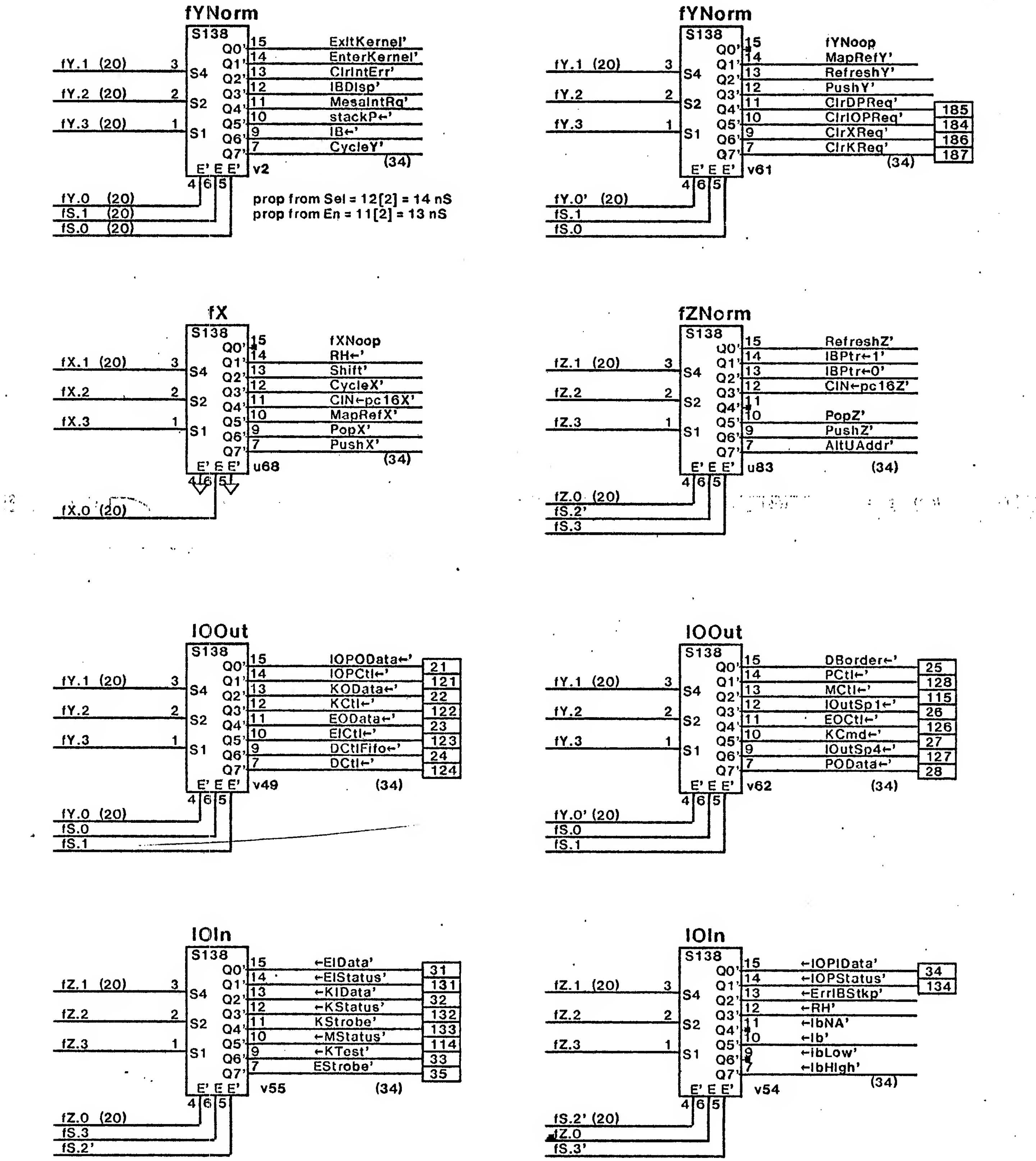
See stackP timings for ErrIBPtr

ErrIBPtr

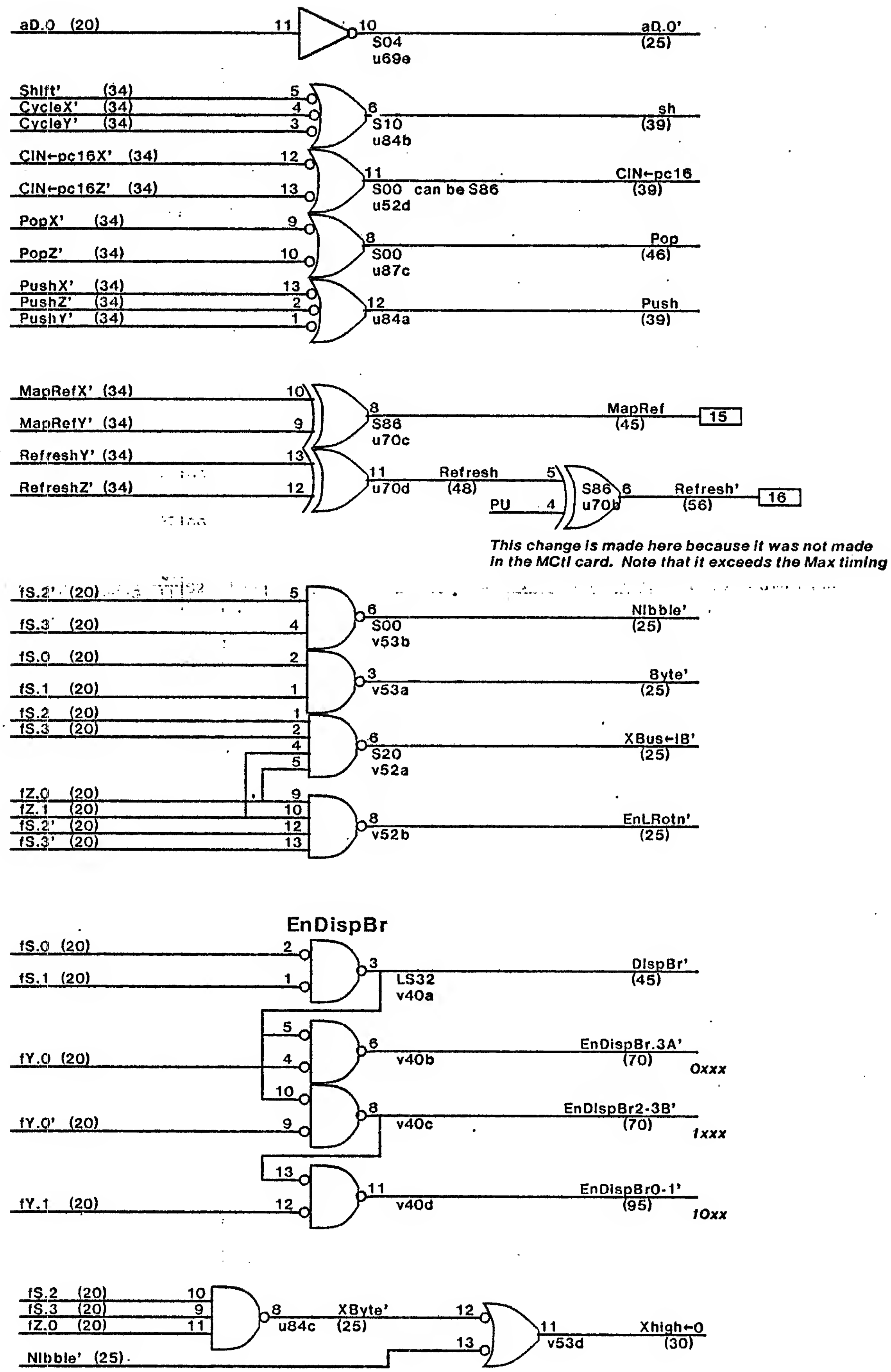




XEROX SDD	Project Dandelion	MIR	File pLionHead08.sil	Designer Garner	Rev J	Date 8/24/80	Page 08



XEROX SDD	Project Dandelion	MIR Decoding I	File pLionHead09.sil	Designer Garner	Rev J	Date 8/23/80	Page 09



XEROX SDD	Project		File	Designer	Rev	Date	Page
	Dandelion	MIR Decoding II	pLionHead10.sil	Garner	J	8/24/80	10

DispBr[0-1] = max(c+32,69,133)

20 ↑ to fY
24[3] S151 select to DispBr
18 DispBr' setup
64[3] = 69

95 ↑ to EnDispBr0-1'
18[2] S151 E' to DispBr
18 DispBr' setup
131[2] = 133 nS

c condition source
12[2] S151 data to DispBr
18 DispBr' setup
c+30[2] = c+32

DsipBr setup

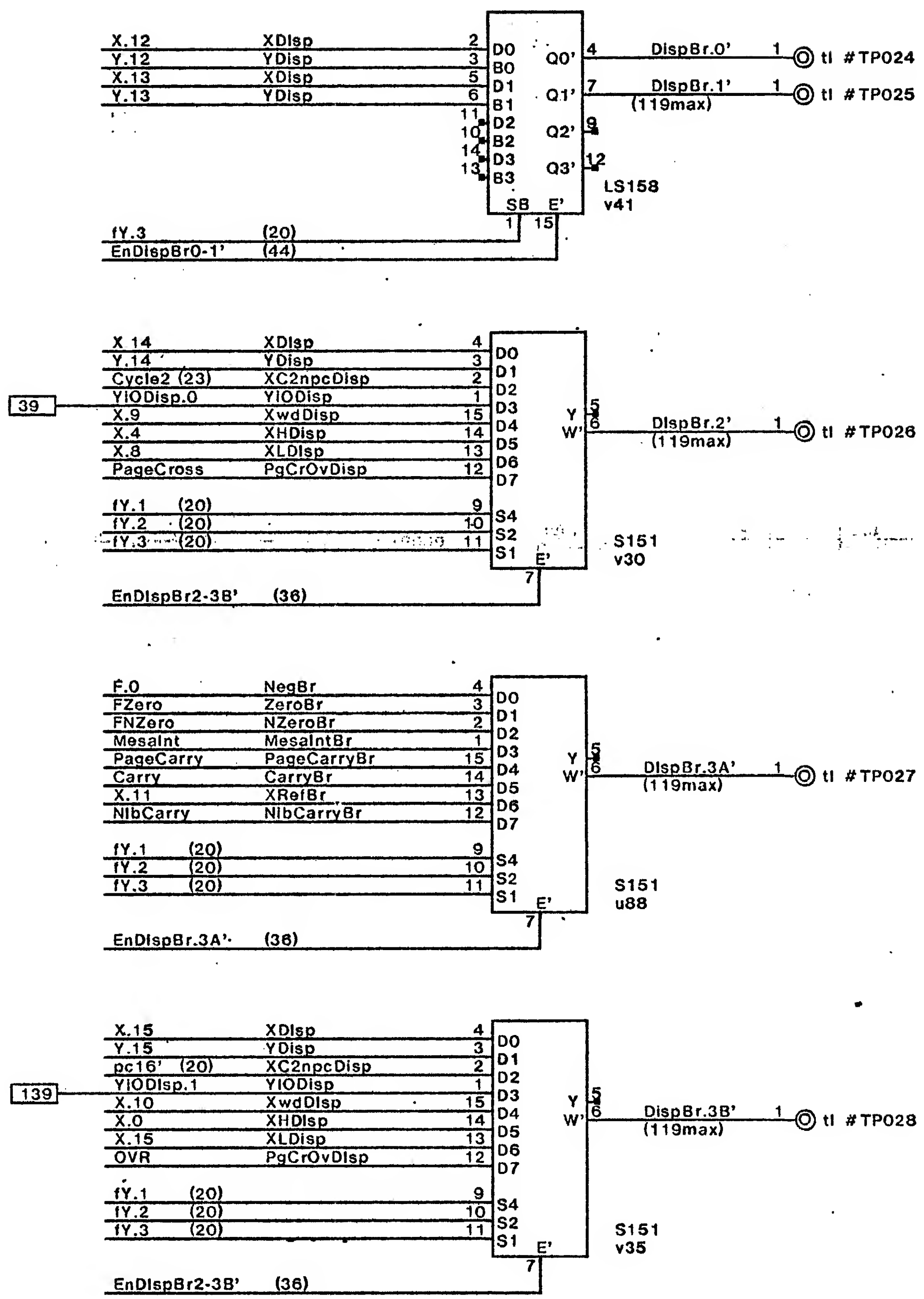
5 S00 in to pTC
6[1] S64 in to pNIA
5[1] 25S09/S374 setup
18 nS

DispBr[2-3] = max(c+26,55,103)

20 ↑ to fY
15[2] S151 select to DispBr
18 DispBr' setup
51[4] = 55 nS

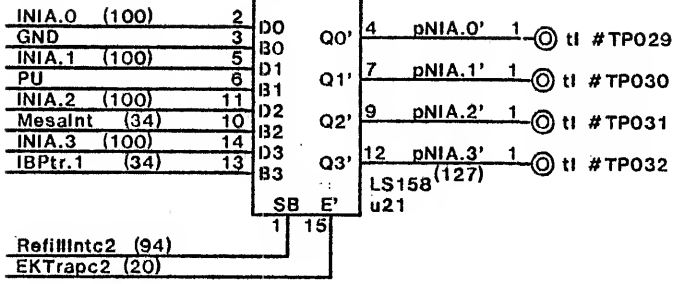
70 ↑ to EnDispBr.3A'
13[2] S151 E' to DispBr
18 DispBr' setup
101[2] = 103 nS

c condition source
7[1] S151 data to DispBr
18 DispBr' setup
c+23[3] = c+26 nS

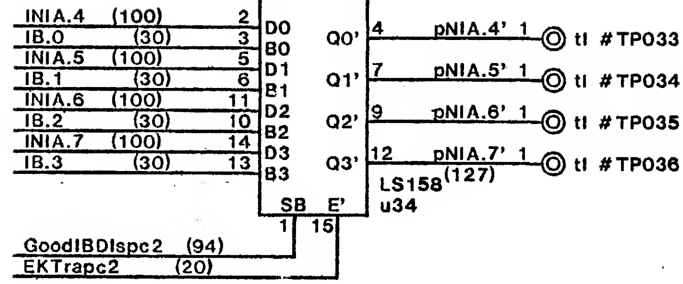


XEROX SDD	*Project* Dandelion	Dispatch/Branch	*File* pLionHead11.sil	*Designer* Garner	*Rev* J	*Date* 8/23/80	*Page* 11

pNIA[0-3]



pNIA[4-7]

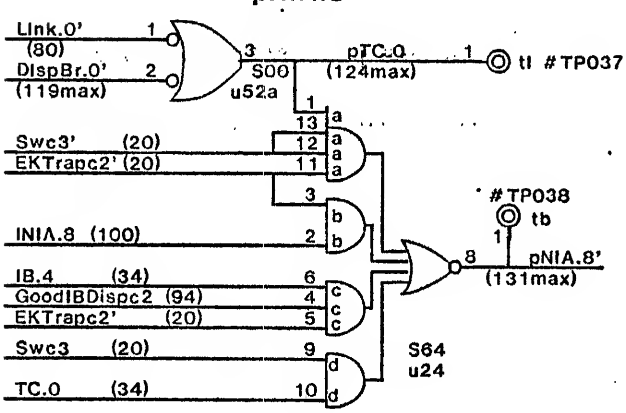


pNIA[0-7] = max(127, 120, 46) nS

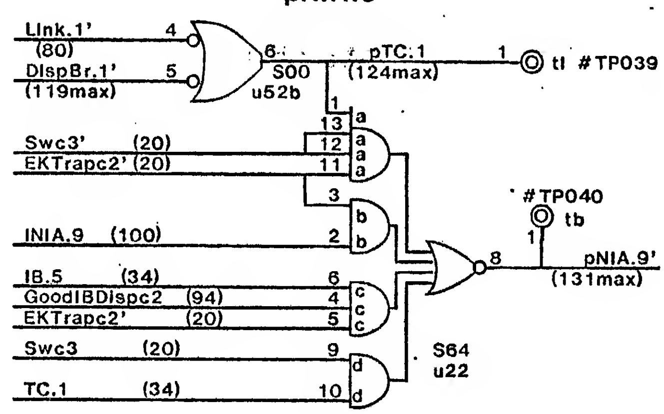
94	*↑ to RefillIntc2*	*100*	*↑ to INIA*	*20*	*↑ to EKErrc2*
24[3]	*LS158 SB to pNIA'*	*12[2]*	*LS158 data to pNIA'*	*18[2]*	*LS158 E' to pNIA'*
5[1]	*25S09/S374 setup*	*5[1]*	*25S09/S374 setup*	*5[1]*	*25S09/S374 setup*
123[4] = 127 nS		*117[3] = 120 nS*		*43[3] = 46nS*	

(See page 11 for pNIA[8-11] timing)

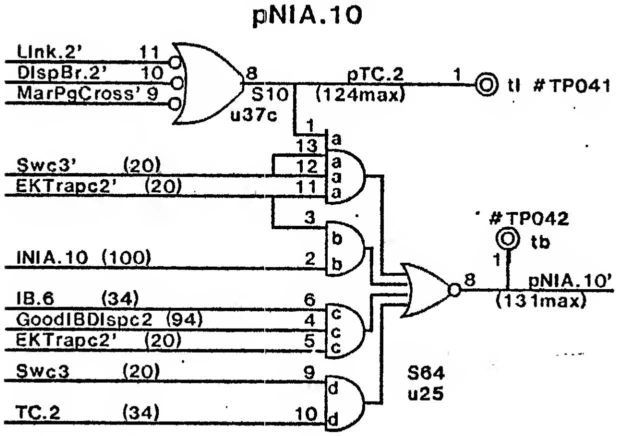
pNIA.8



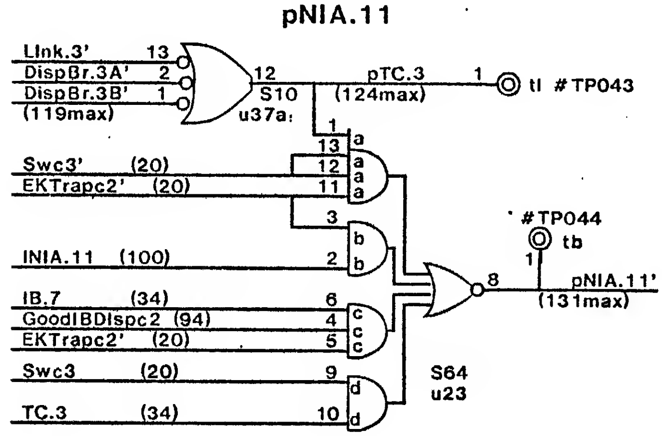
pNIA.9



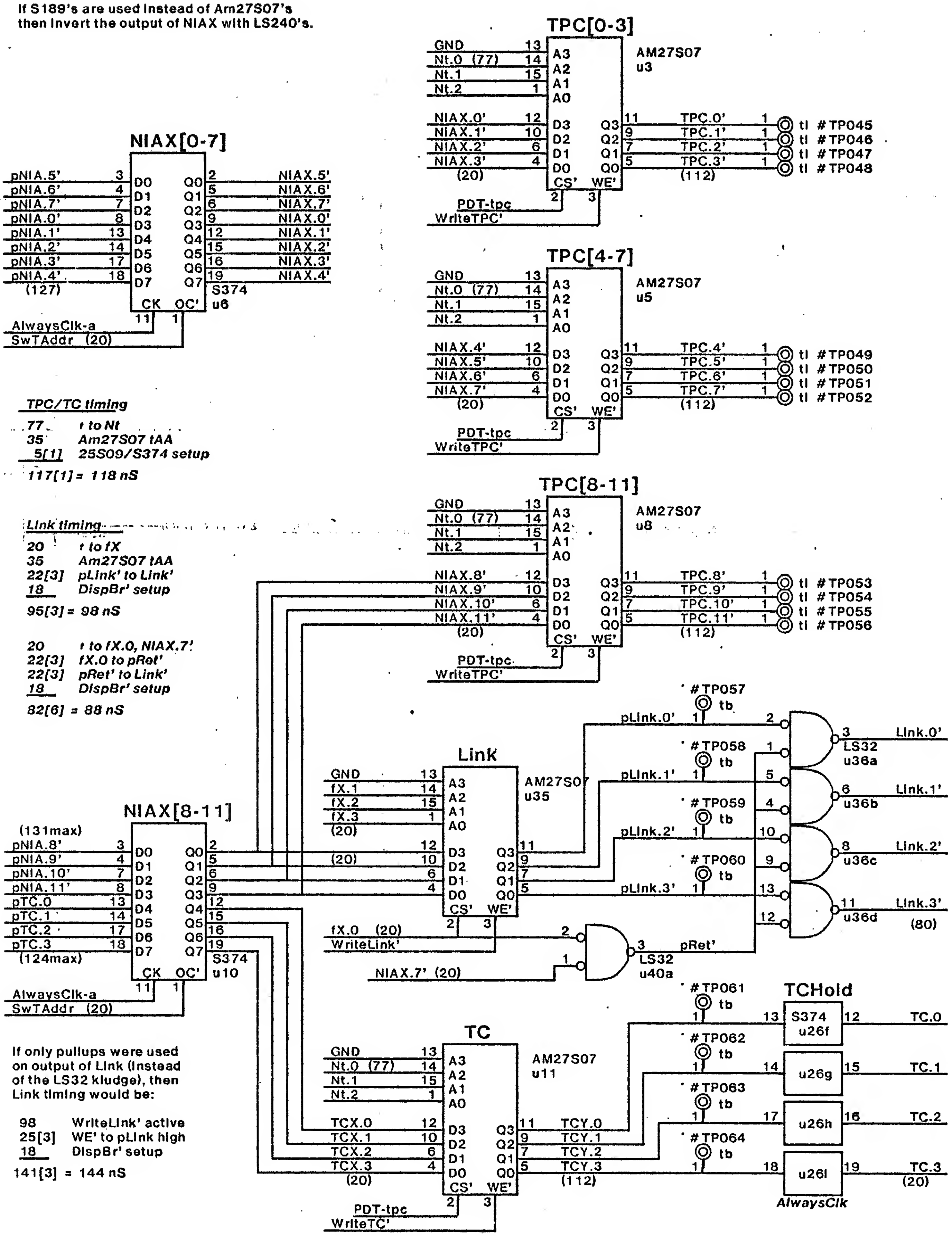
pNIA.10



pNIA.11



If S189's are used instead of Am27S07's
then invert the output of NIAX with LS240's.



TPC/TC timing

77	*t to Nt*
35	*Am27S07 tAA*
5[1]	*25S09/S374 setup*
117[1] = 118 nS	

Link timing

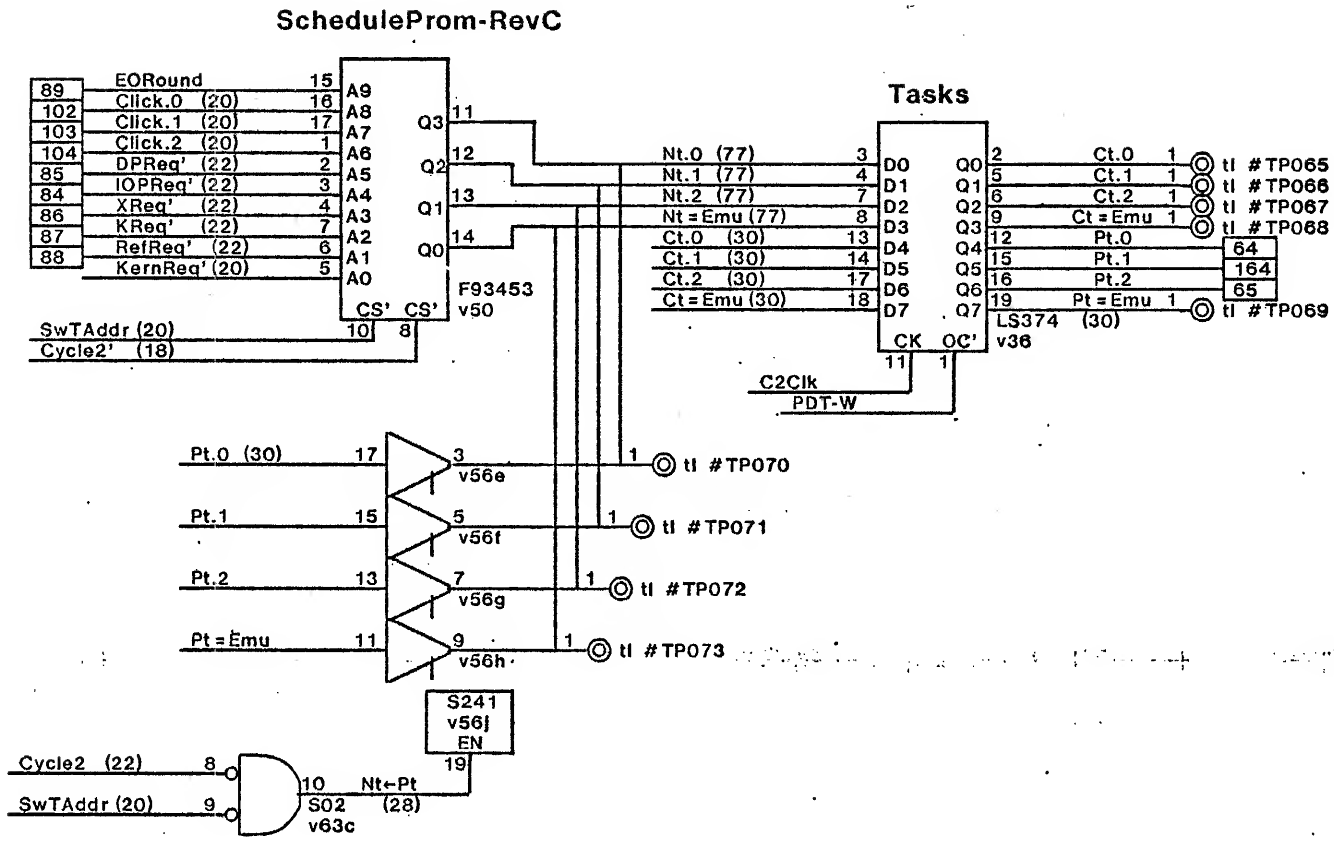
20	*t to fX*
35	*Am27S07 tAA*
22[3]	*pLink' to Link'*
18	*DispBr' setup*
95[3] = 98 nS	

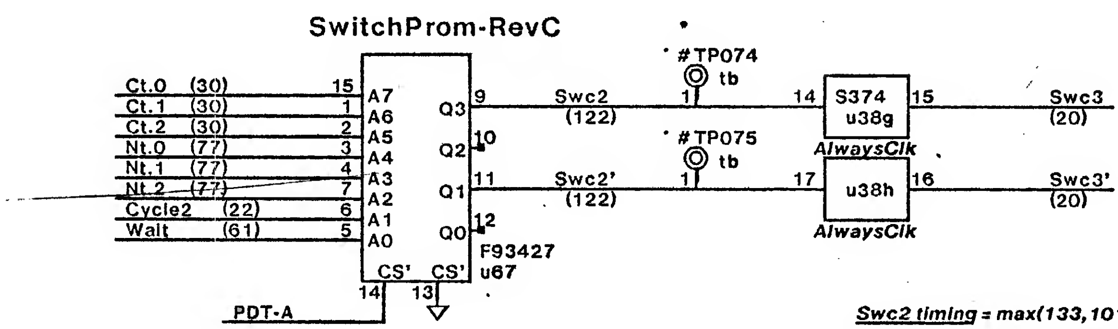
20	*t to fX.0, NIAX.7'*
22[3]	*fX.0 to pRet'*
22[3]	*pRet' to Link'*
18	*DispBr' setup*
82[6] = 88 nS	

If only pullups were used on output of Link (instead of the LS32 kludge), then Link timing would be:

98	WriteLink' active
25[3]	WE' to pLink high
18	DispBr' setup
141[3] = 144 nS	

XEROX SDD	Project: Dandelion	TPC, TC, & Link	File: pLionHead13.sil	Designer: Garner	Rev: J	Date: 8/23/80	Page: 13





	Nt (Prom)	*Nt*	*Ct*	*Pt*
c1	*3-S*	*Previous*	*Current*	*Previous*
c2	*Next*	*Next*	*Current*	*Previous*
c3	*3-S*	*Current*	*Next*	*Current*

Swc2 timing ***= max(133,101,101)***

22 ***↑ to Kreq'***
55 ***F93453 addr to Nt***
45 ***F93427 addr to Swc2***
10[1] ***25S09 SB setup***

132[1] = 133 nS

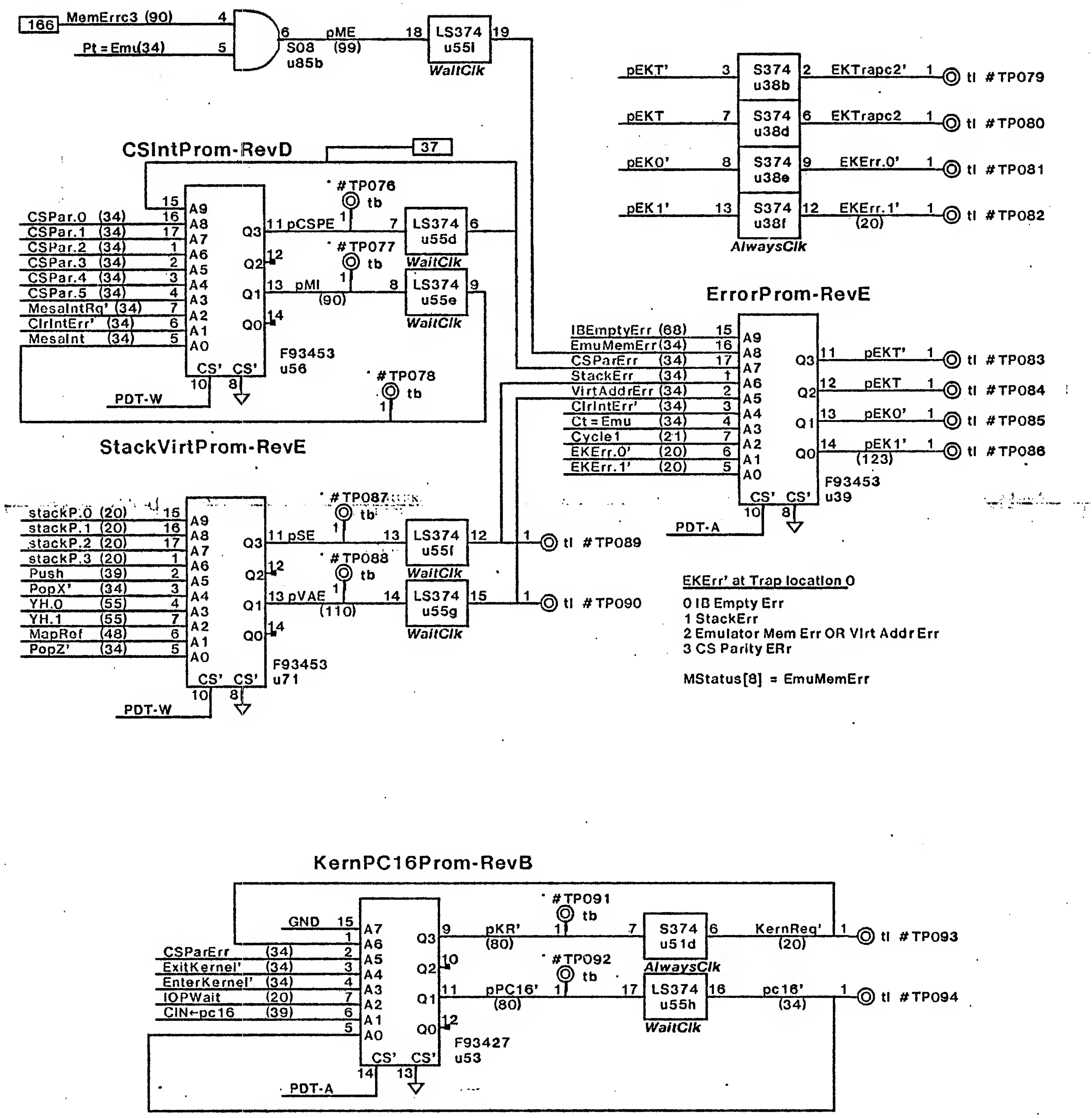
20 ***↑ to SwTAddr***
25 ***F93453 CS' to Nt***
45 ***F93427 addr to Swc2***
10[1] ***25S09 SB setup***

100[1] = 101 nS

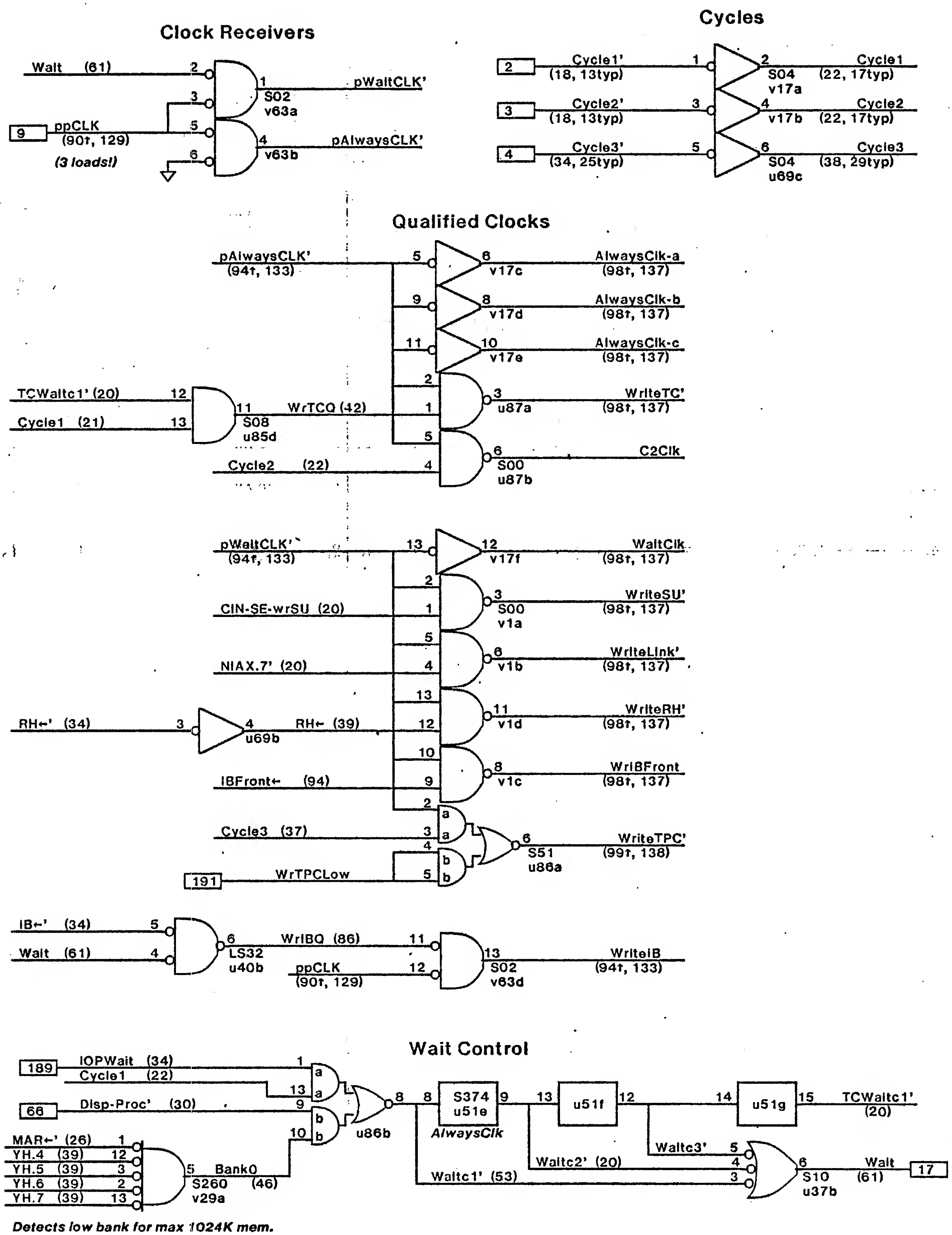
28 ***↑ to Nt←Pt***
15[2] ***S241 EN to Nt***
45 ***F93427 addr to Swc2***
10[1] ***25S09 SB setup***

98[3] = 101 nS

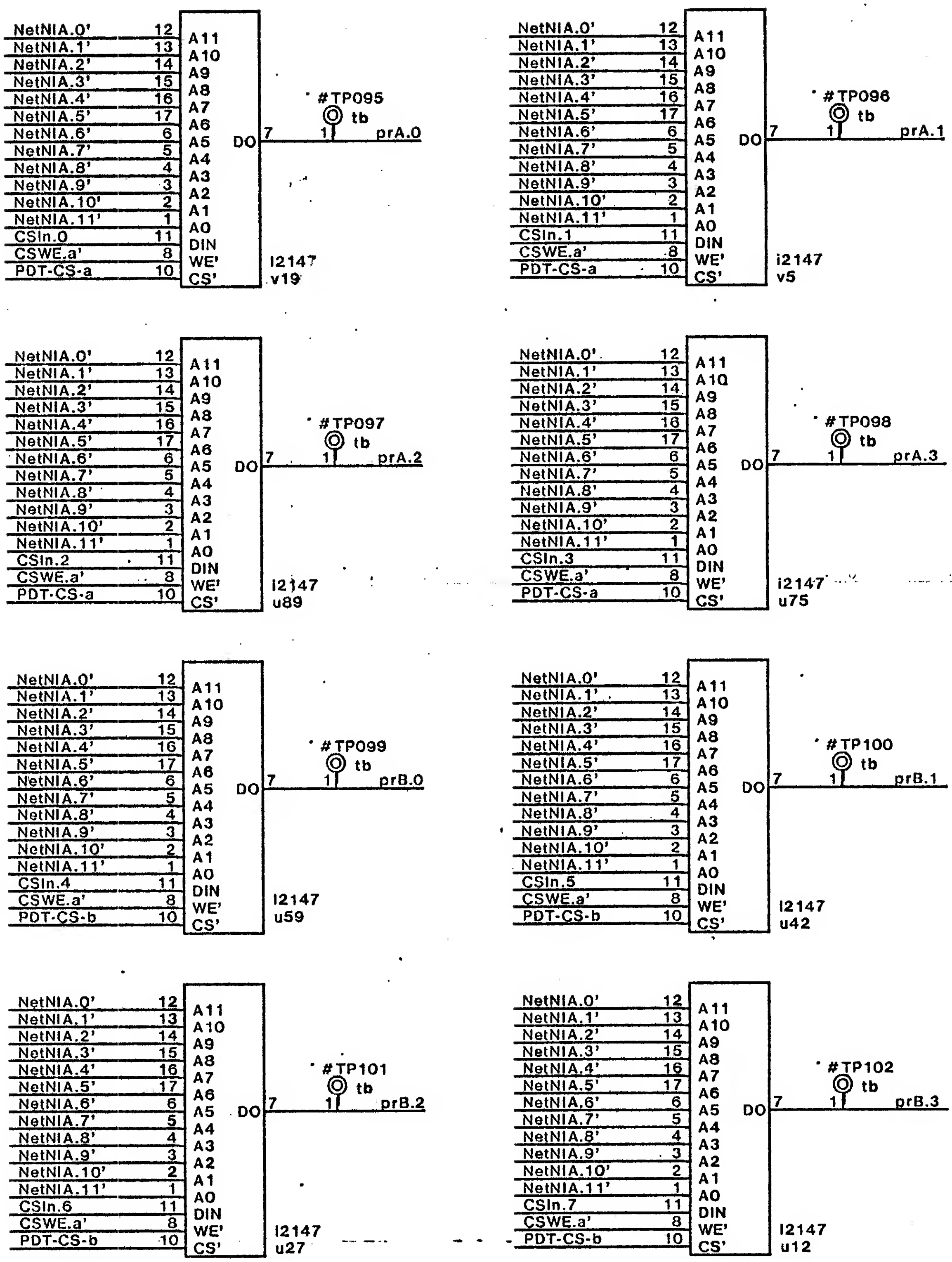
XEROX SDD	*Project* Dandelion	Schedule, Switch, & Tasks	*File* pLionHead14.sil	*Designer* Garner	*Rev* J	*Date* 8/23/80	*Page* 14



XEROX SDD	Project		File	Designer	Rev	Date	Page
	Dandelion	Error, Emulator, & Kernel Proms	pLionHead15.sil	Garner	J	5/14/80	15



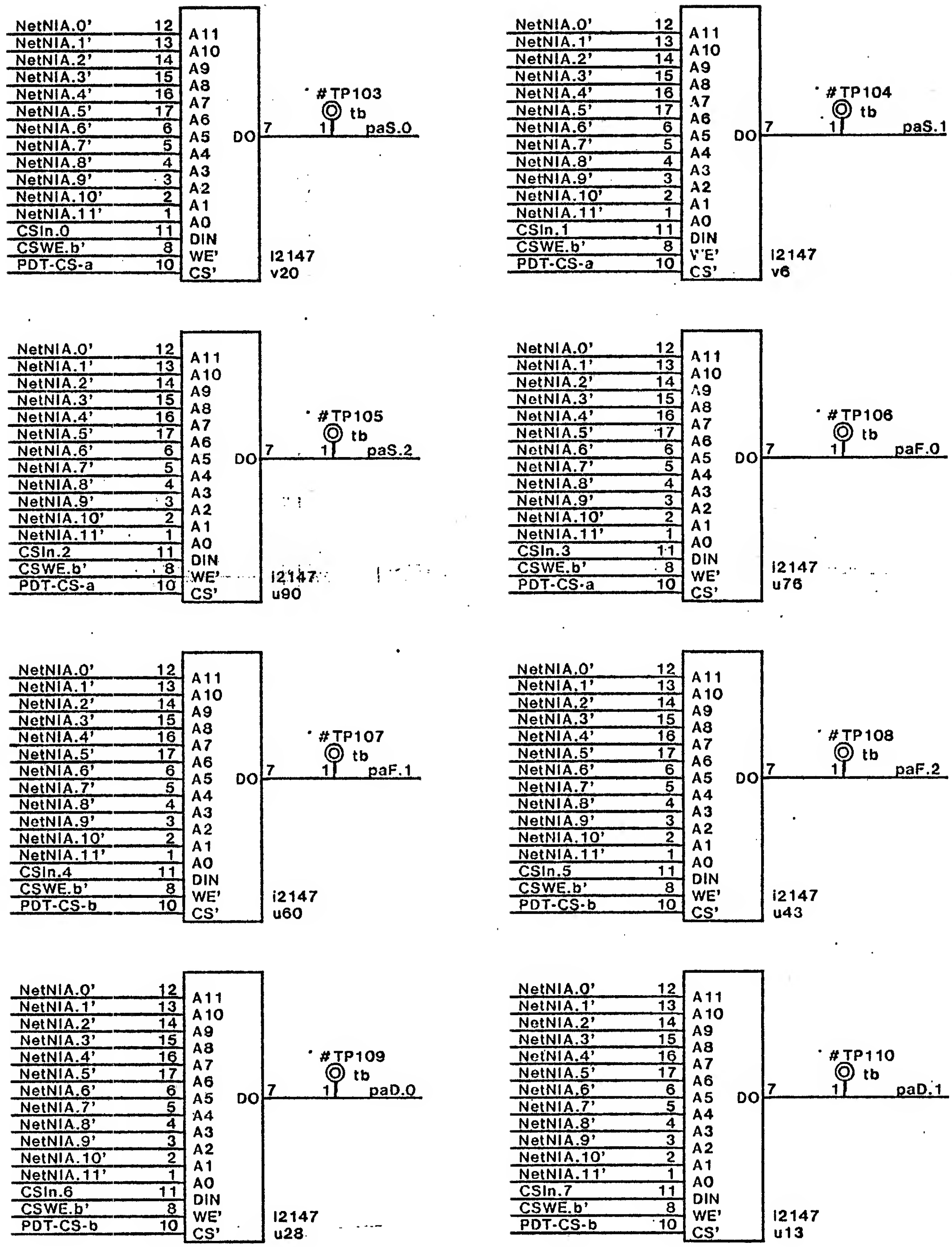
XEROX SDD	Project Dandelion	Clocks, Wait	File pLionHead16.sil	Designer Garner	Rev J	Date 8/23/80	Page 16



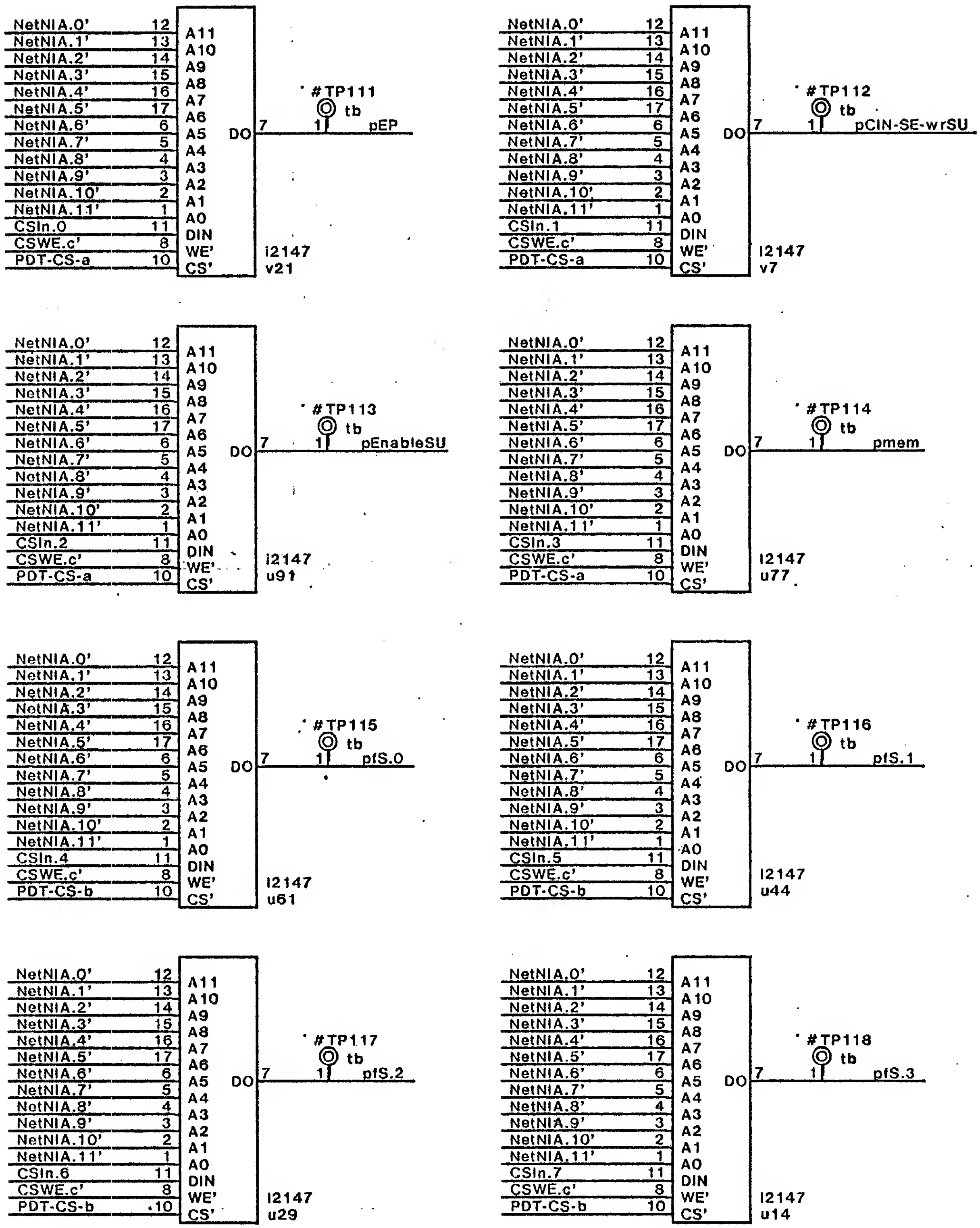
CS Timing

17	*t to NIA'*
13	*transmission delay*
70	*tAA 2147L*
100 nS	

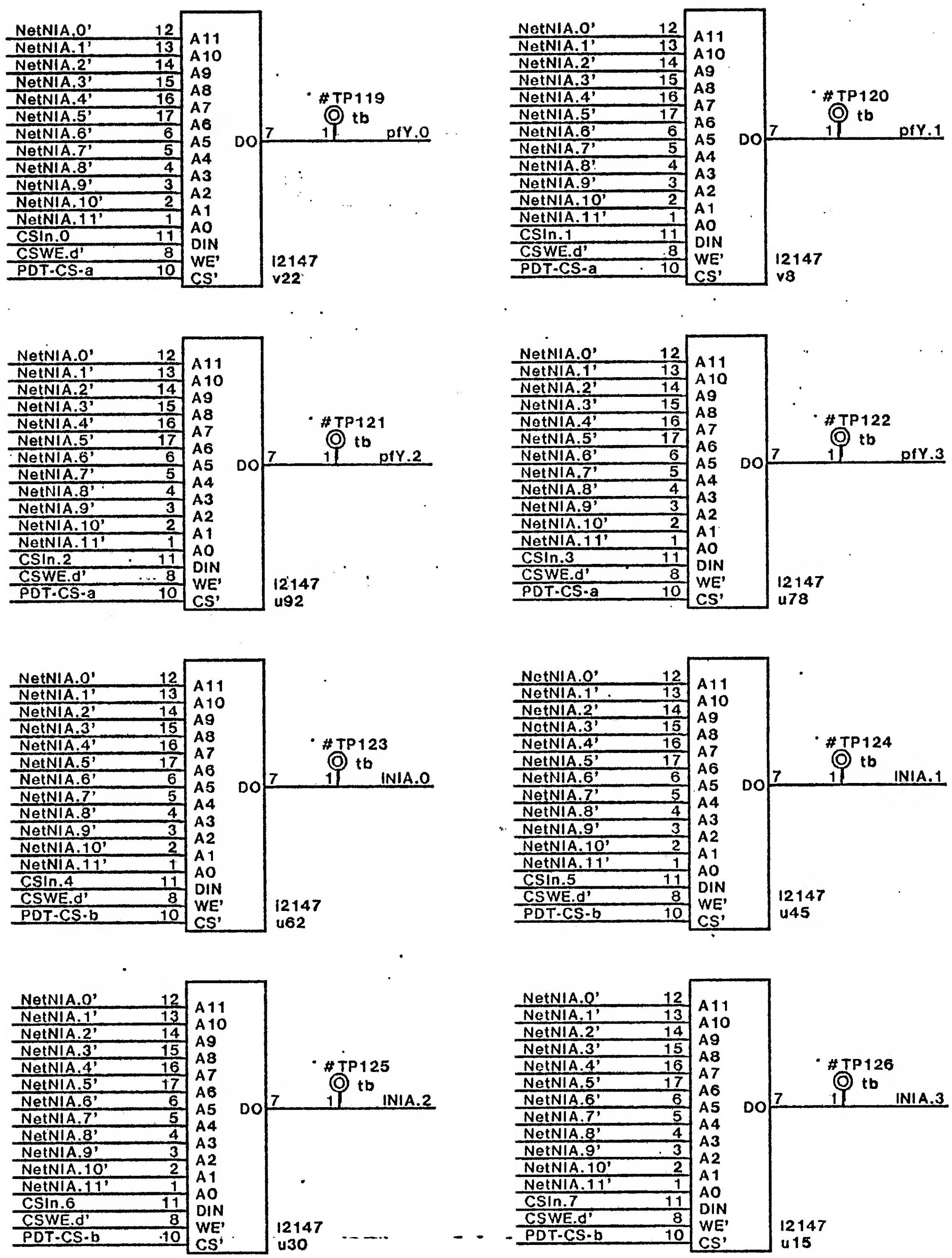
XEROX SDD	*Project* Dandelion	Control Store A [0-7]	*File* pLionHead17.sil	*Designer* Garner	*Rev* J	*Date* 5/14/80	*Page* 17



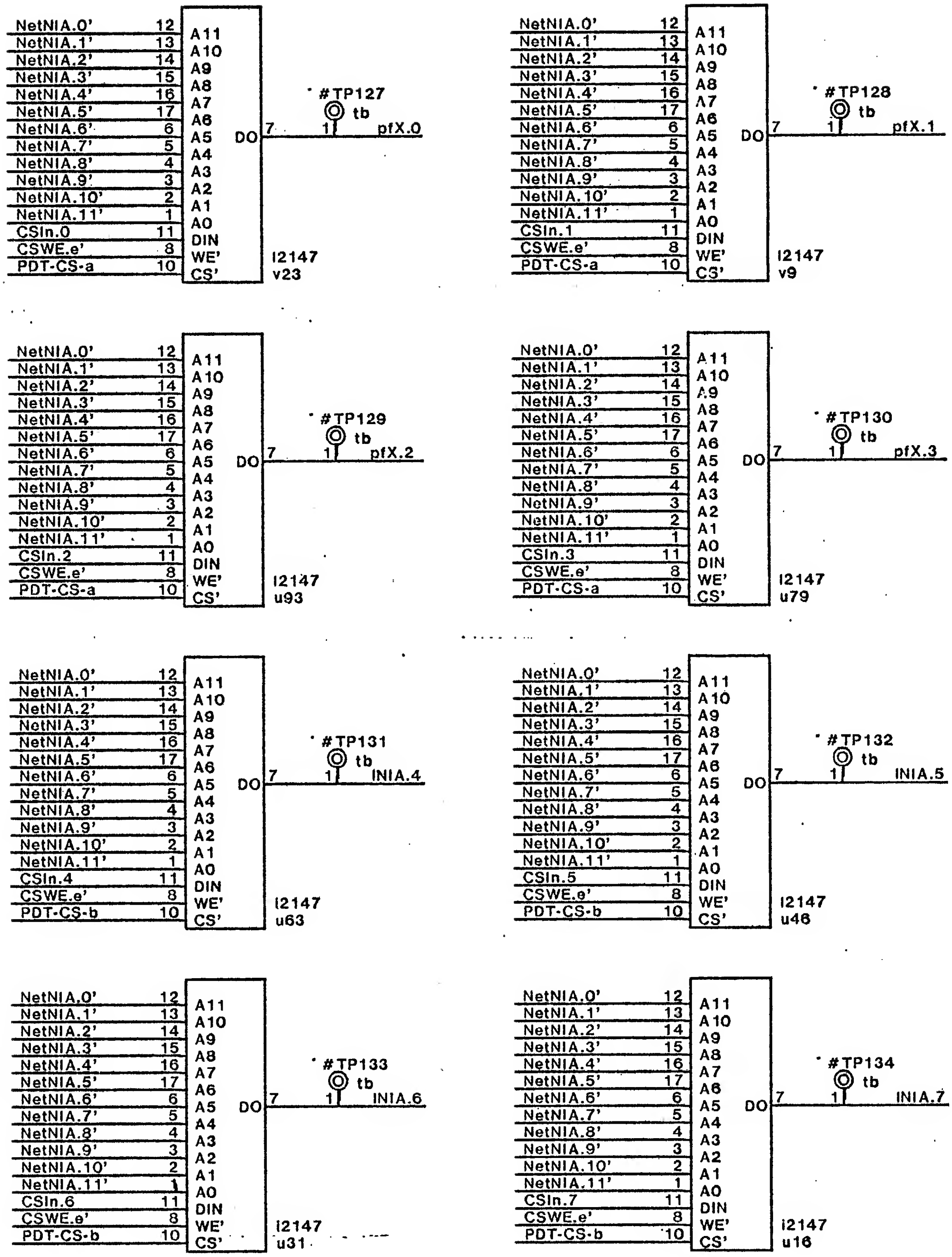
XEROX SDD	Project		File	Designer	Rev	Date	Page
	Dandelion	Control Store B [8-15]	pLionHead18.sil	Garner	J	5/14/80	18



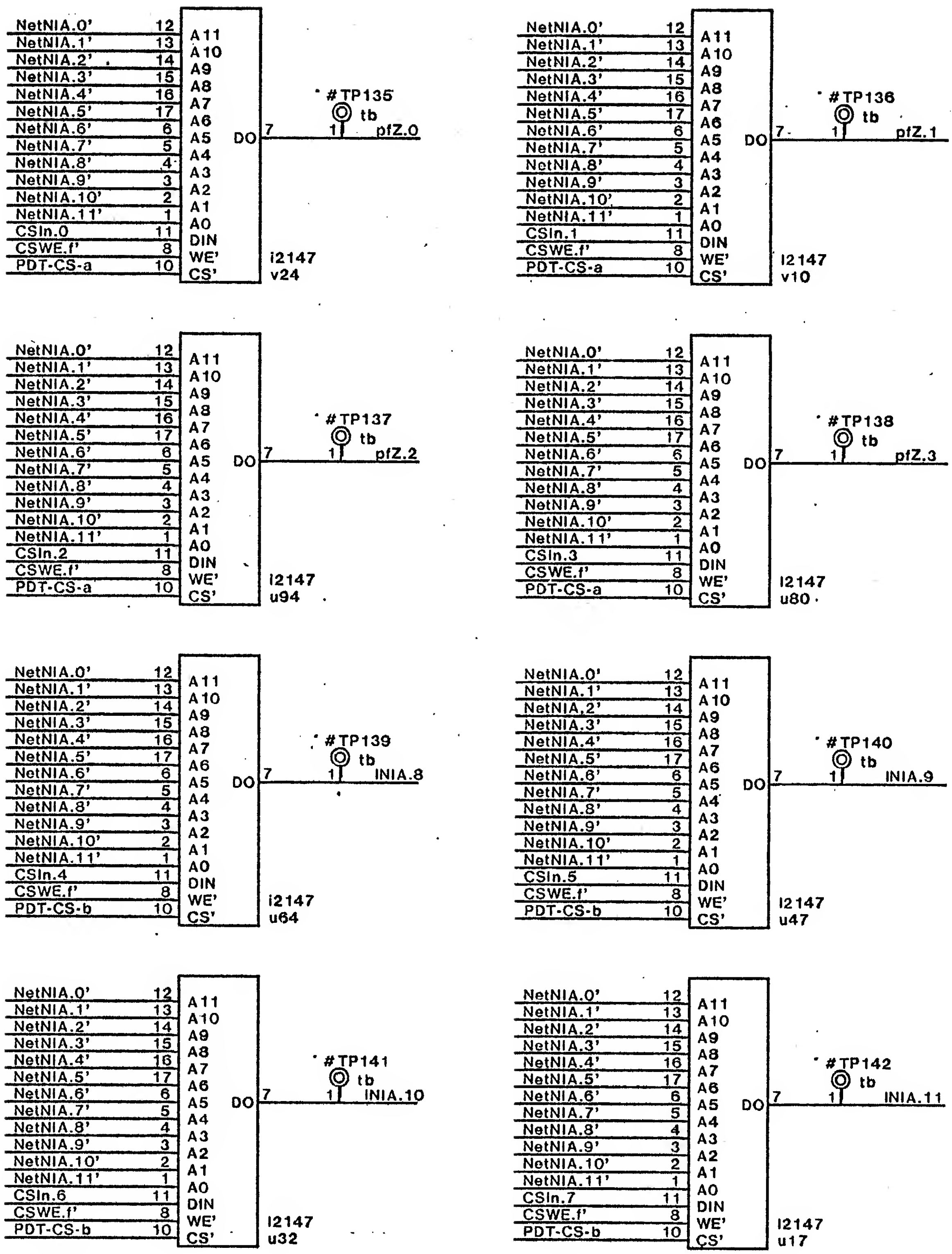
XEROX SDD	Project		File	Designer	Rev	Date	Page
	Dandelion	Control Store C [16-23]	pLionHead19.sil	Garner	J	5/14/80	19



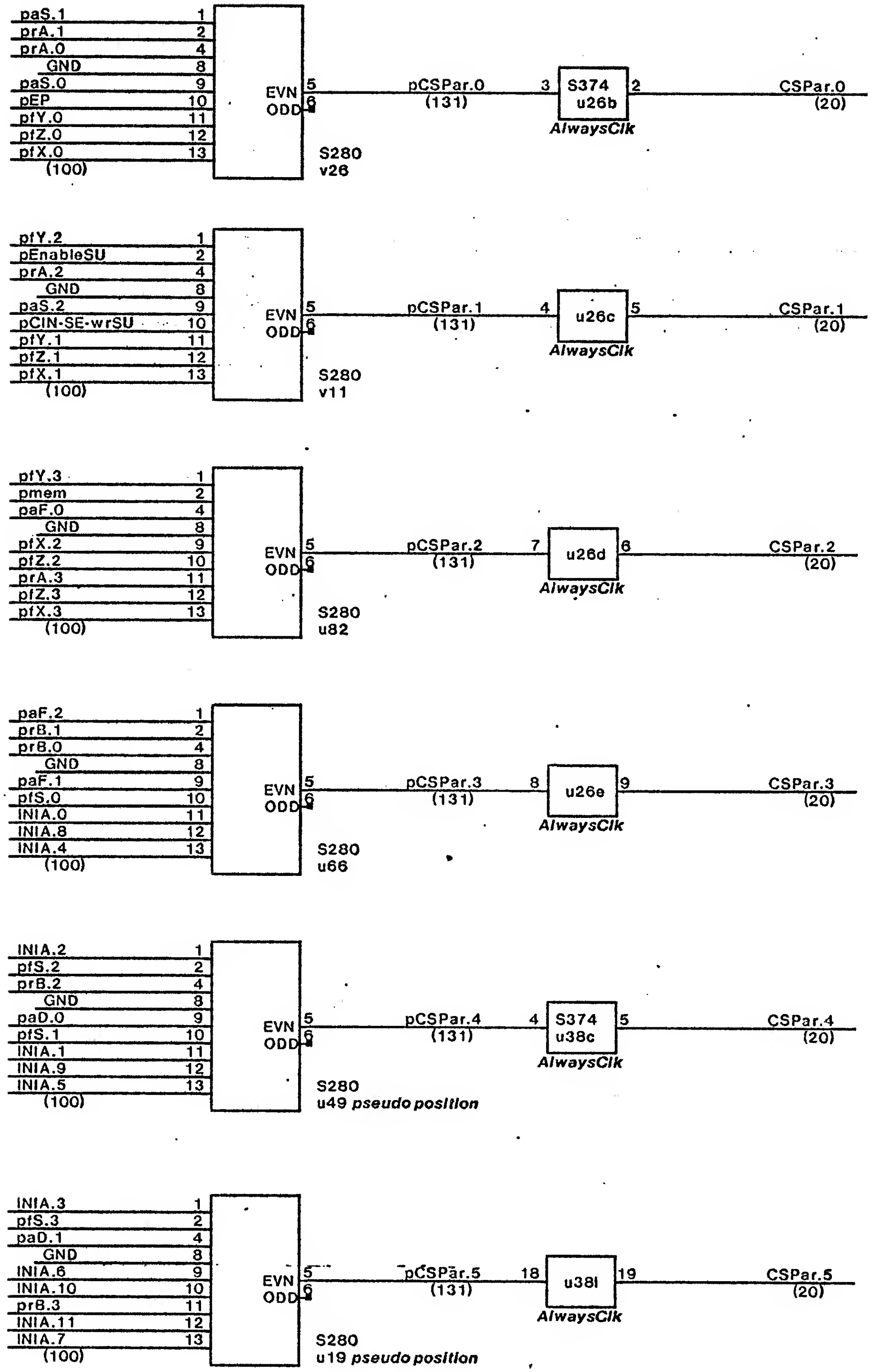
XEROX SDD	Project		File	Designer	Rev	Date	Page
	Dandelion	Control Store D [24-31]	pLionHead20.sil	Garner	J	5/14/80	20



XEROX SDD	Project Dandelion	Control Store E [32-39]	File pLionHead21.sil	Designer Garner	Rev J	Date 5/14/80	Page 21

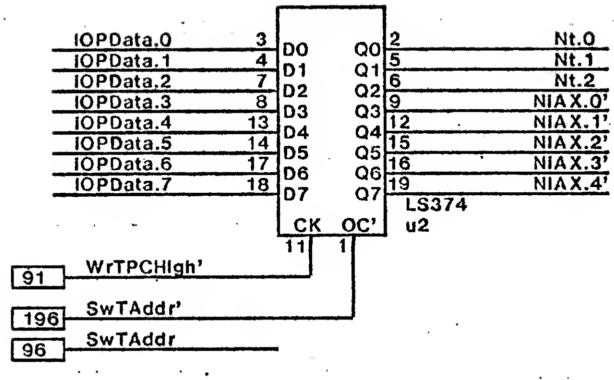


XEROX SDD	Project Dandelion	Control Store F [40-47]	File pLionHead22.sil	Designer Garner	Rev J	Date 8/23/80	Page 22

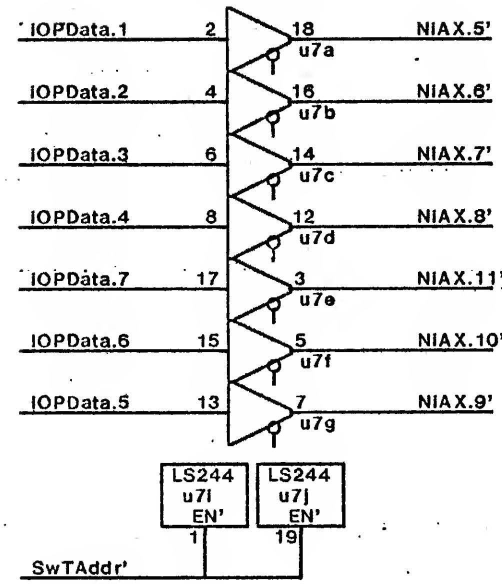


XEROX SDD	Project		File	Designer	Rev	Date	Page
	Dandelion	CS Parity (PC)	pLionHead23.sil	Garner	J	5/13/80	23

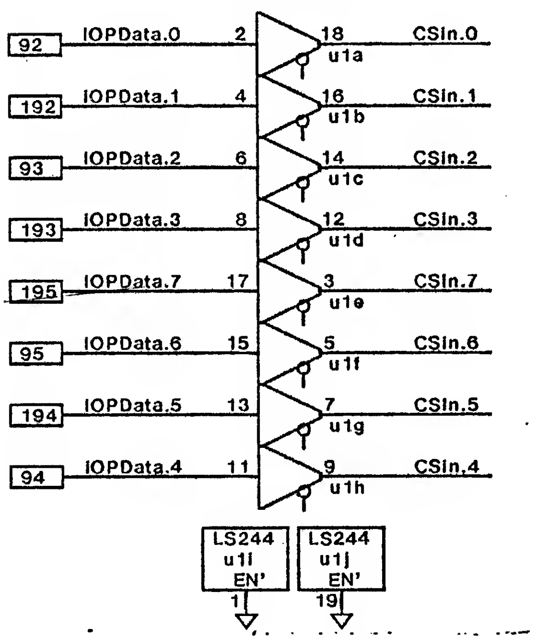
IOP TPCHigh



IOP TPCLow

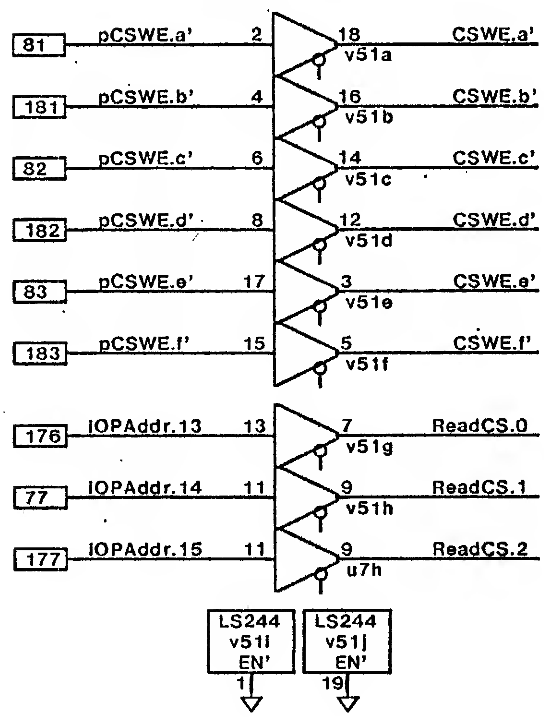


CS Data In

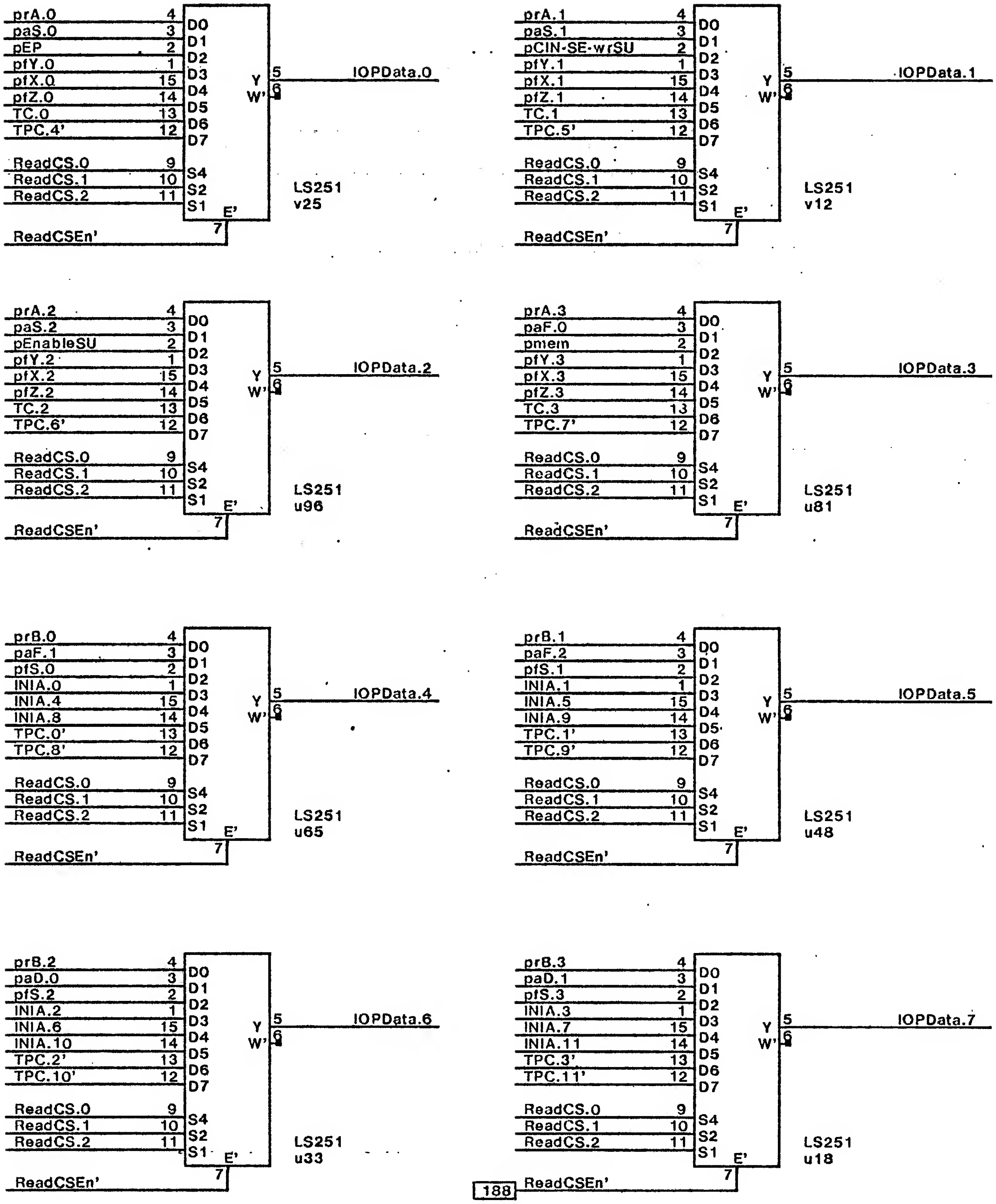


For LS241 pullup pin 19.

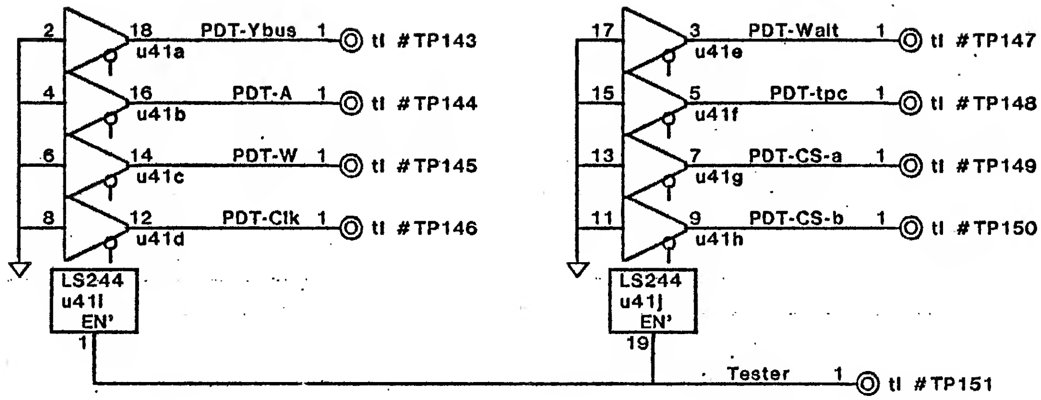
CSWE, IOPAddr Recv



XEROX SDD	Project Dandelion	Tasks, IOP TPC-TC Control	File pLionHead24.sil	Designer Garner	Rev J	Date 8/23/80	Page 24



XEROX SDD	Project		File	Designer	Rev	Date	Page
	Dandelion	CS Read	pLionHead25.silGarner		J	8/23/80	25



The Control Store can be read & written via backplane pins. Once tested, instructions (or parts of instructions), can be loaded in order to test additional features. For instance, all X-bus sources can be disabled by loading a 6 into CS bits 16-23 (controlled by CSWE.c'). Simple programs to test the 2901's can also be executed in this way.

The SU & RH registers can be loaded by controlling EnableSU, CIN-SE-wrSU, & RH← from a microinstruction.

stackP, IB, High SU Addr, & Low SU Addr can be similarly tested.

The MIR & MIR decoding can be tested by loading instructions into the CS.

PDT-Ybus is used to test devices attached to the Y bus.

PDT-A is used to disable registers or Proms whose outputs go to a register clocked by AlwaysClk.

PDT-W is similarly used for WaitClk.

PDT-Clk & PDT-Wait disable the outputs of AlwaysClk & WaitClk'd registers.

The following steps cause a CS byte to be written. It is assumed that the TPC has been written with the required CS address.

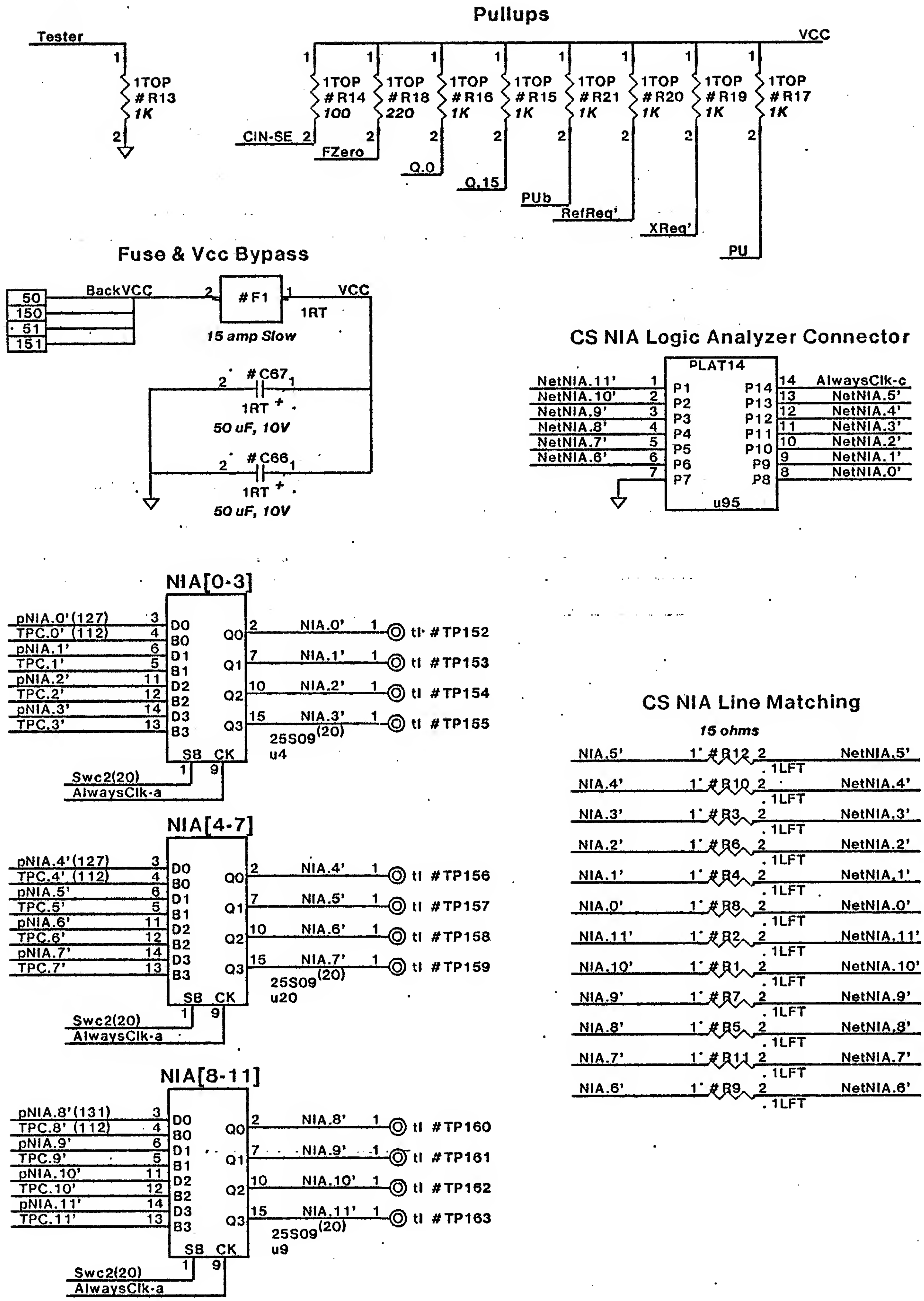
```
PDT-Clk ← 1; Swc3←1;              {cause NIA to come from TPC}
IOPWait ← 1;
SwTAddr' ← 0; SwTAddr ← 1;        {init code}
    IOPData ← data
    CSWE.x' ← 0; CSWE.x' ← 1;
```

If IOPWait is left high, the CP will not execute the instruction which has been loaded into the CS. Instead, the CP will be frozen in a state where the instruction is totally decoded, but the result will no be loaded into any register. Thus, all the microinstruction register (MIR) decoding logic can be tested without even executing an instruction.

The following steps cause the TPC to be written:

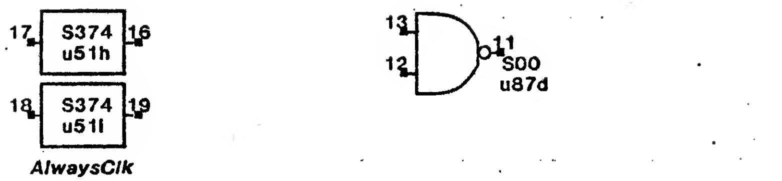
```
IOPWait ← 1;                                        {init code}
SwTAddr' ← 0; SwTAddr ← 1;
    IOPData ← (addr lshift5) or (data rshift7);     {set TPC addr & high 5 bits of data}
    WrTPCHigh' ← 0; WrTPCHigh' ← 1;
    IOPData ← data and 7F'x;
    WrTPCLow ← 0; WrTPCLow ← 1;                     {write low 7 bits}
```

D0 card test programs for reading & writing TPC & CS available on [Iris]<Workstation>LH>CardTest.dm



XEROX SDD	Project		File	Designer	Rev	Date	Page
	Dandelion	PC Discretes & NIA	pLionHead27.s	il Garner	J	8/24/80	27

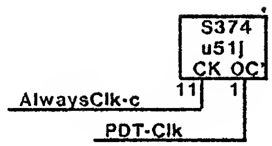
Unused Parts



Junk 374 Allocation

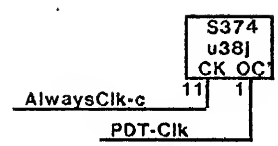
S374
g15 - Always Clock

b	*MAR←*
c	*AllowMDR←*
d	*KernReq'*
e	*WaitC2*
f	*WaitC3*
g	*TCWait*
h	
i	



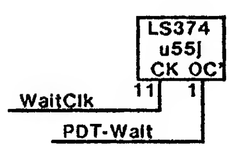
S374
h16 - Always Clock

b	*EKTrapc2'*	
c	*CSPar.4*	PC only
d	*EKTrapc2*	
e	*EKErr.0'*	
f	*EKErr.1'*	
g	*Swc3*	
h	*Swc3'*	
i	*CSPar.5*	PC only



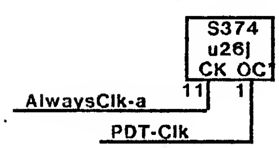
LS374
h17 - Wait Clock

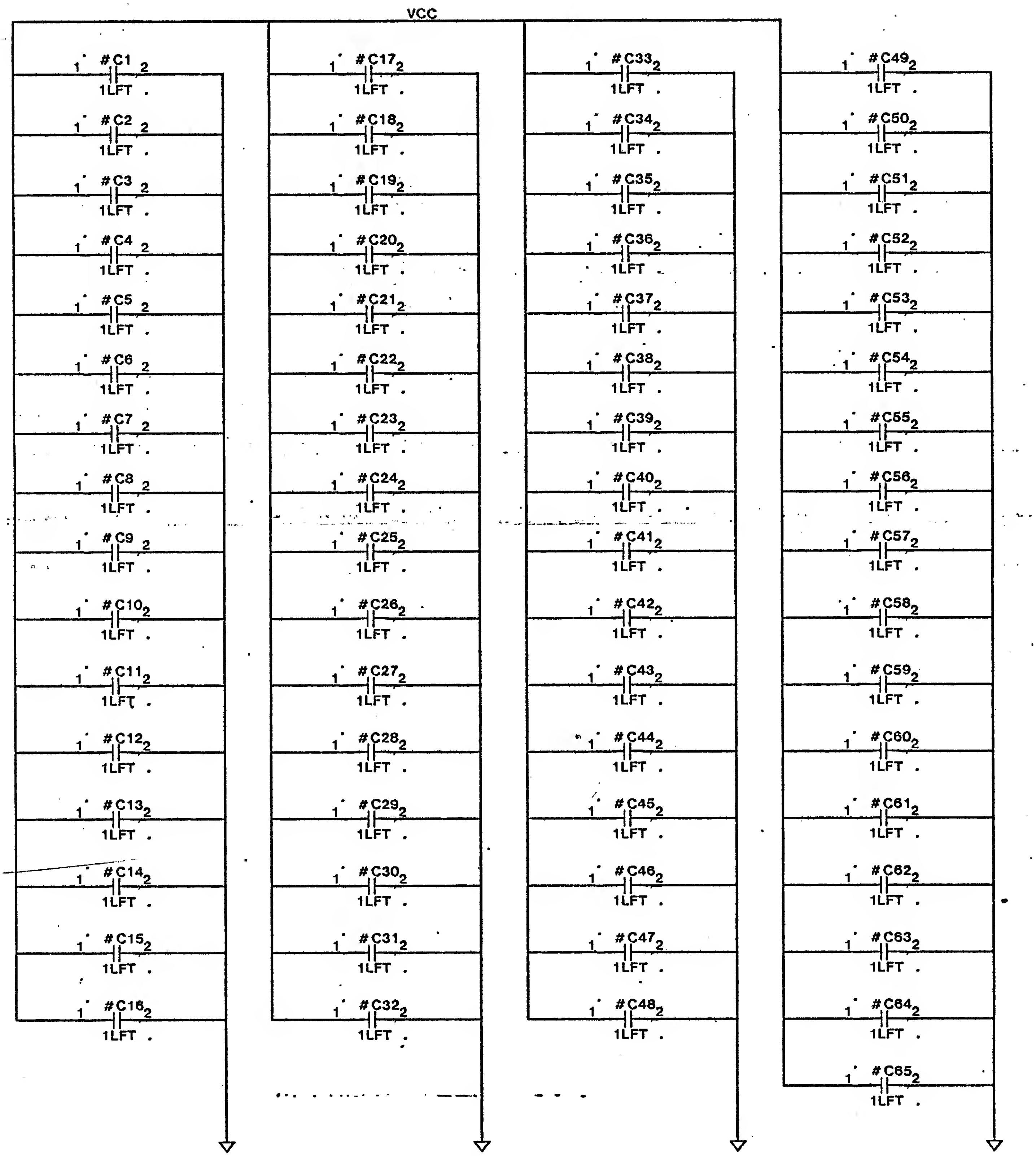
b	*IBPtr.0*
c	*IBPtr.1*
d	*CSParErr*
e	*MesaInt*
f	*StackErr*
g	*VirtAddrErrc2*
h	*pc16'*
i	*MemErrc3*



S374
f9 - Always Clock

b	*CSPar.0*
c	*CSPar.1*
d	*CSPar.2*
e	*CSPar.3*
f	*TC.0*
g	*TC.1*
h	*TC.2*
i	*TC.3*





NOTE: C1-C65, CAP., CERAM, 50V, .10UF, PART NO. 702W05218

XEROX ED	Project		File	Designer	Rev	Date	Page
	Dandelion	Filter Capacitors	pLionHead29.sil	Lin	J	8/23/80	29

pLion head-M.Pres

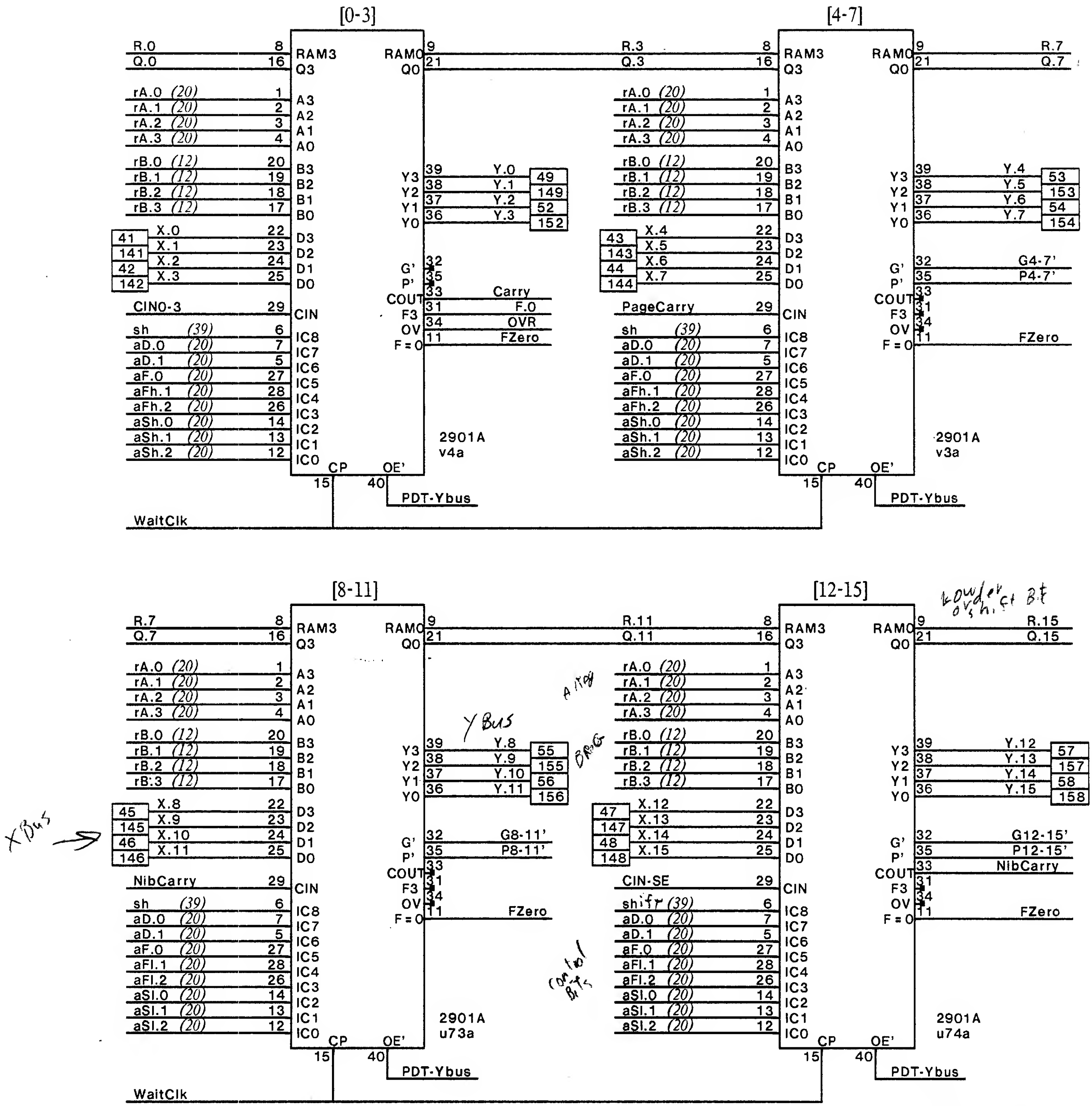
Dandelion Central Processor (CP)

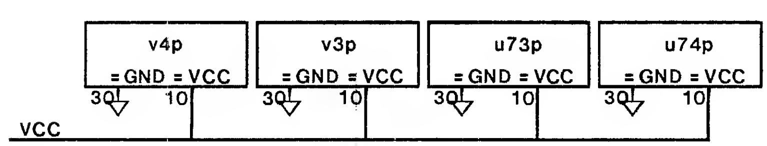
pLionHead# #.sil are the printed circuit board schematics.
sLionHead# #.sil are the stichweld board schematics.
LionHead# #.sily are documentation pages.

Also see:

[Iris]<Workstation>LH>#LionHead-M.press	-- s or p schematics
[Iris]<Workstation>LH>CPProms-K.dm	--Proms
[Iris]<Workstation>LH>DMR.press	--Dandelion Microcode Reference
[Iris]<Workstation>LH>CPCheckOut.press	
[Iris]<Workstation>LH>DLionIORules.press	--Rules for IO controllers

XEROX SDD	Project Dandelion	Contents	File LionHead00.sily	Designer Garner	Rev M	Date 4/2/81	Page 0



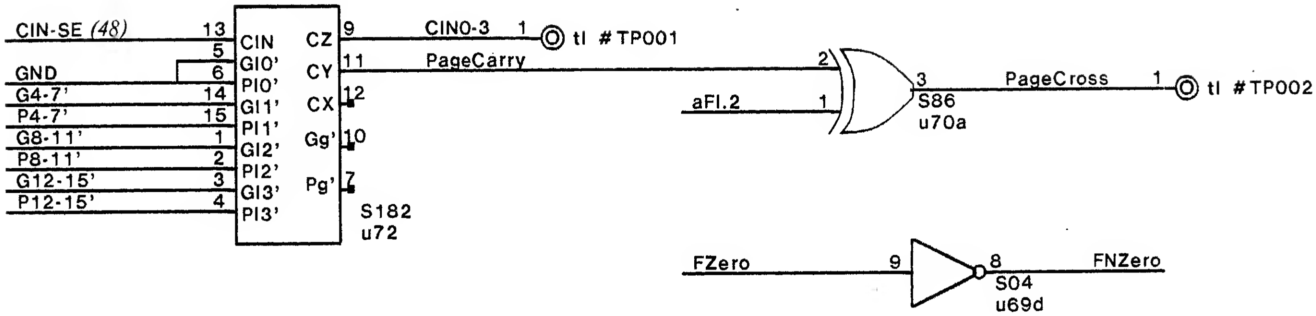


sh,.aD setup = 54 nS
aF setup = 45 nS
aS setup = 45 nS

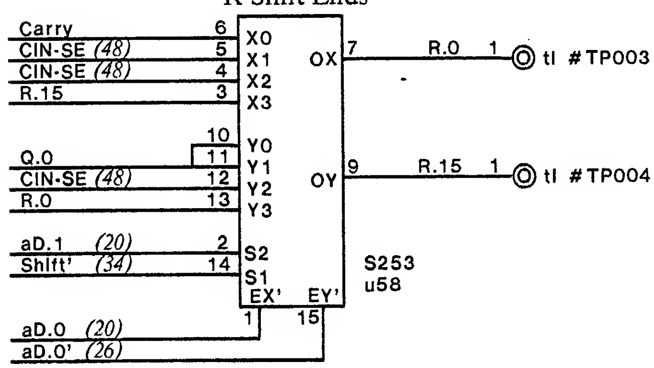
rA	*to G,P =*	*45*	*rA*	*to Y =*	*50 nS*
D	*to G,P =*	*30*	*D*	*to Y =*	*32*
			Cn	*to Y =*	*25*
rA	*to Cout =*	*50*	*aS*	*to Y =*	*40*
D	*to Cout =*	*32*	*aF*	*to Y =*	*35*
Cin	*to Cout =*	*16*	*aD*	*to Y =*	*25*

XEROX SDD	Project Dandelion	2901 chips	File pLionHead01.sil	Designer Garner	Rev M	Date 8/23/80	Page 01

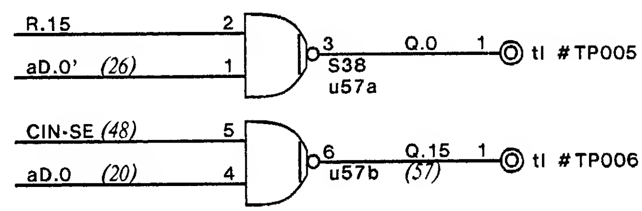
2901 Carry
Lookahead



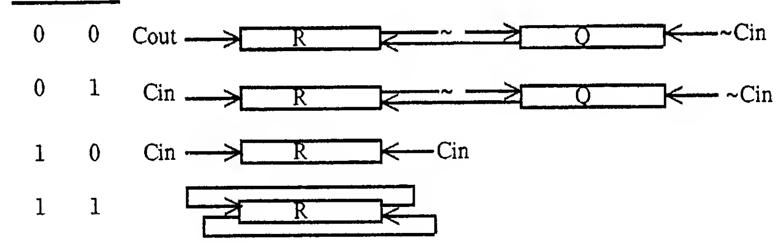
R Shift Ends



Q Shift Ends



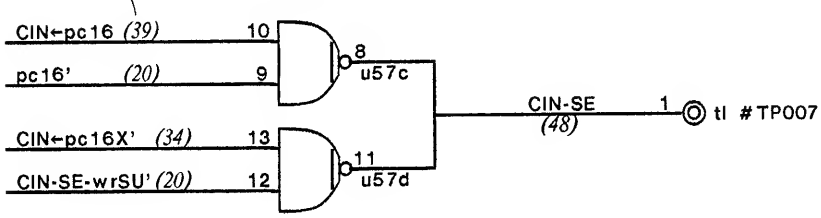
aD.1 shift'



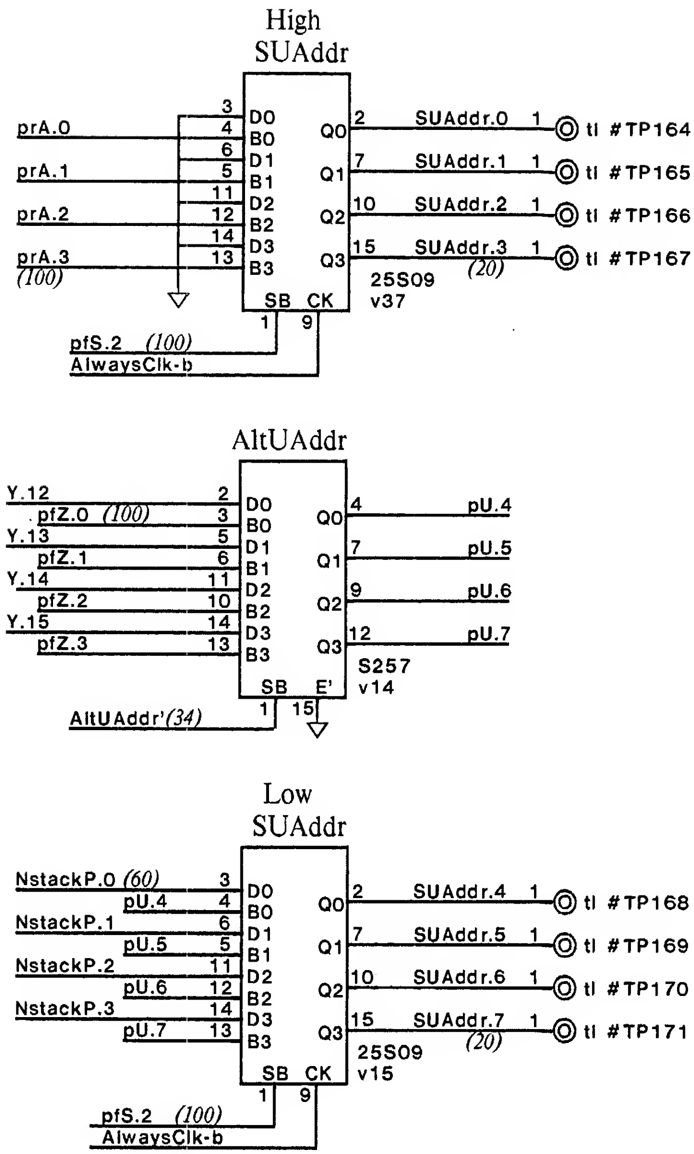
aD.0 = 0 implies right shift

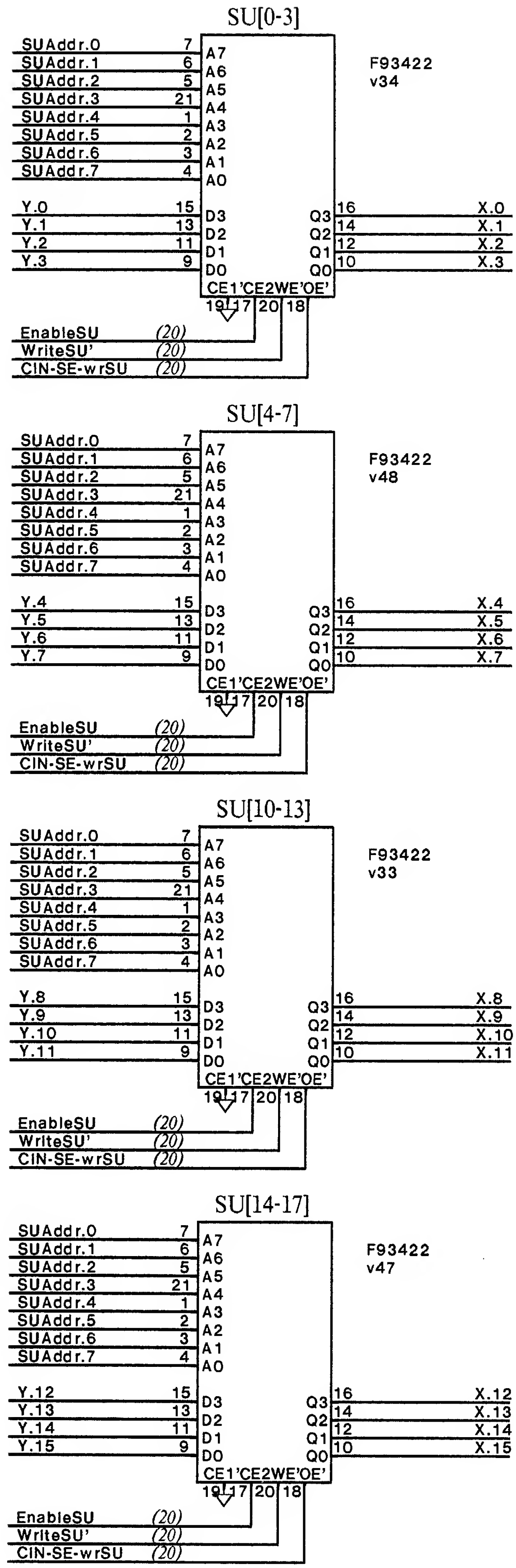
timing IV0.5 measure ↑ worst case
ALWAYS <K ns

Cin & Shift Ends



XEROX SDD	Project Dandelion	Lookahead, Shift Ends, Cin	File pLionHead02.sil	Designer Garner	Rev M	Date 8/23/80	Page 02





SU X-bus disable

15[3]	↑ to CIN-SE-wrSU (tPLH)
30	Output Disable
10	X-bus

55[3] = 58 nS

XBus ← SU = max(75,60) nS

17[3]	↑ to SUAddr
45	tAA
10	X-bus

72[3] = 75 nS

17[3]	↑ to CIN-SE-wrSU/EnableSU
30	F93422 OE'/CE2 to X-bus
10	X-bus

57[3]=60 nS

SU write setup

5[1]	Data setup
39	WE

44[1] = 45 nS

AltUAddr setup

5[1]	25S09 setup
8[1]	Y -> pU

13[2] = 15 nS (26 if LS257)

F93422 data t-hold = 5 nS

XEROX SDD	Project Dandelion	SU	File pLionHead03.sil	Designer Garner	Rev M	Date 12/4/80	Page 03

RH Setup

7[1] S240
25[3] S189 setup

32[4] = 36 nS

XBus ← RH = max(64, 59) nS

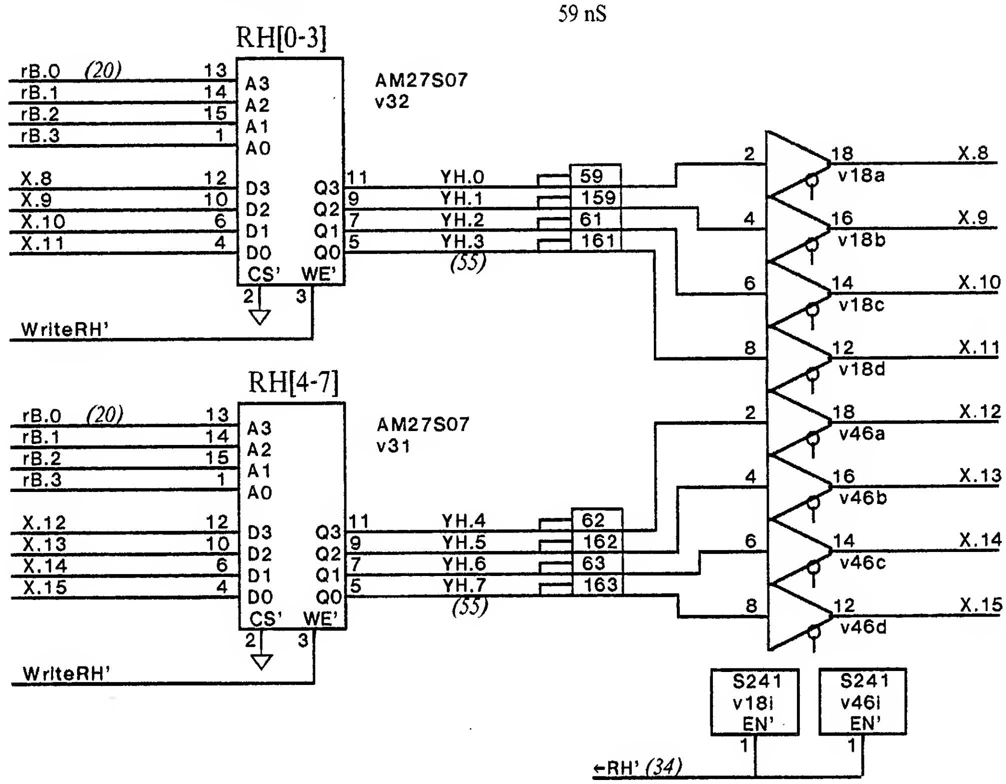
17[3] ↑ to rB'
35 S189 tAA (wr recovery = 35 nS)
9 YH to X
10 X-bus

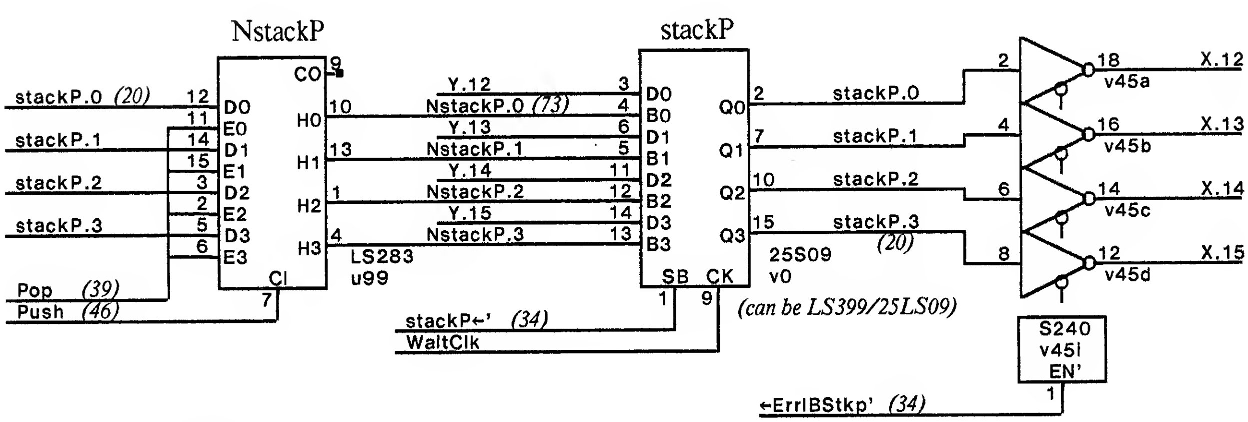
71[3] = 74 nS

34 ↑ to ←RH'
15 S241 EN' to X-bus
10 X-bus

59 nS

If the S189 must be used intead of the 29701, then place S240 between Xbus & S189 inputs.





XBus ← stackP = max(59, 38) nS

17[3] ↑ to stackP
7 S240 data to X-bus
10 X-bus

34[3] = 38 nS

34 ↑ to ←ErrIntstackP'
15 S240 EN' to X-bus
10 X-bus

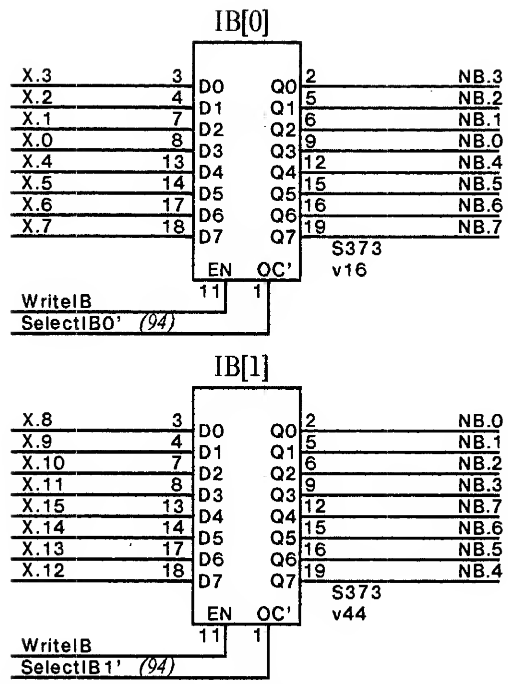
59 nS

push timing

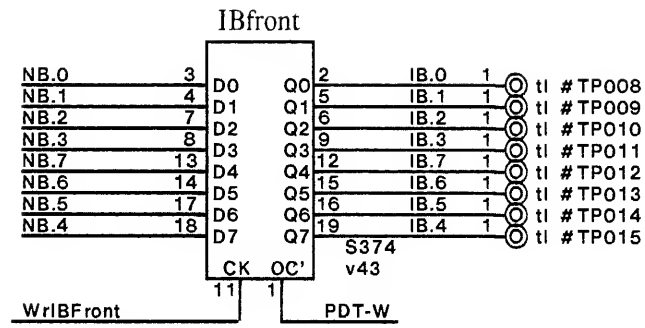
46 ↑ to Push
24[3] Push to NstackP
5[1] 25S09 setup

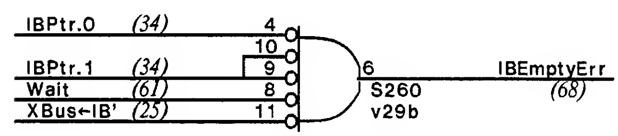
75[4] = 79 nS

XEROX SDD	Project Dandelion	RH, stackP	File pLionHead04.sil	Designer Garner	Rev M	Date 8/24/80	Page 04



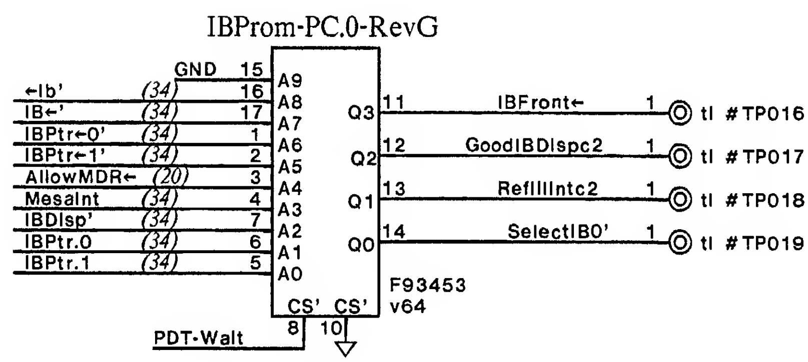
S373 t-hold=10 nS, t-setup=0 nS.
(falling edge of WriteIB occurs 4 nS before rising edge of Clk.)

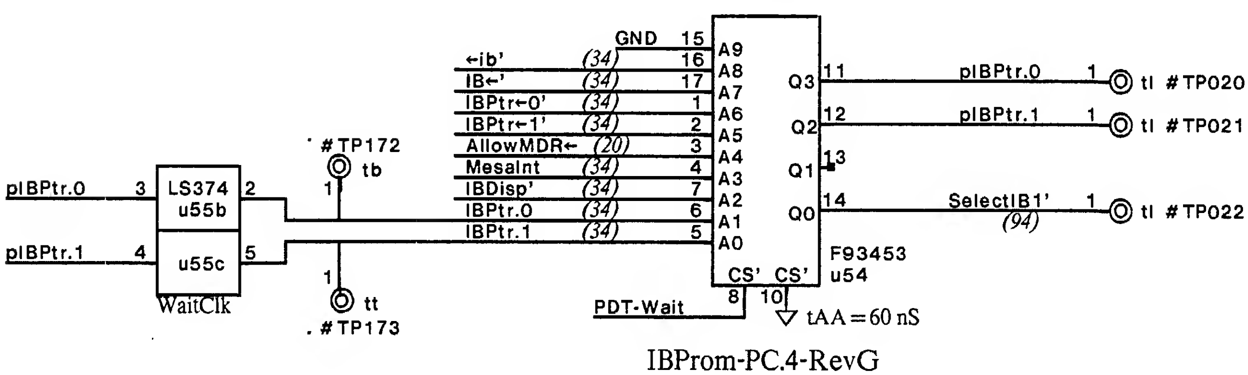




IBfront←IB[1]

34	↑ to IBPtr←1'
60	tAA
18[2]	SelectIB1' to NB
20[2]	LS374 setup
132[4]=136 nS	





Timing for HM7649 IBProm:

IBFront ← Xbus = (x+37, x+36) nS

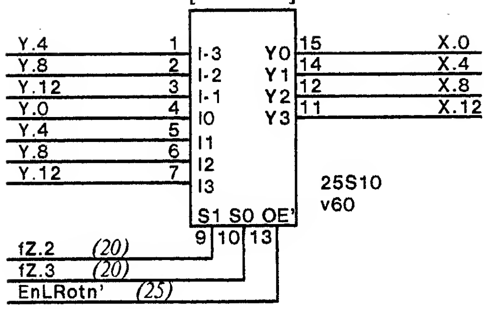
x	Xbus to IB
43	WriteIB rises 43 nS before end of cycle
-6	Difference between S373 "EN to Q" and "Data to Q" = 18[2] - 13[1] = 6 nS. Data can arrive 6 nS after WriteIB goes high.
x+37 nS	

x	Xbus to IB
13[1]	S373 Data to NB
20[2]	LS374 setup
x+36 nS	

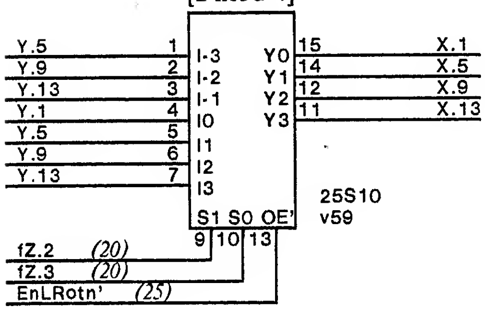
94	WriteIB rises
18[2]	S373 EN to NB
20[2]	LS374 setup
132[4]=136 nS	

LRotn

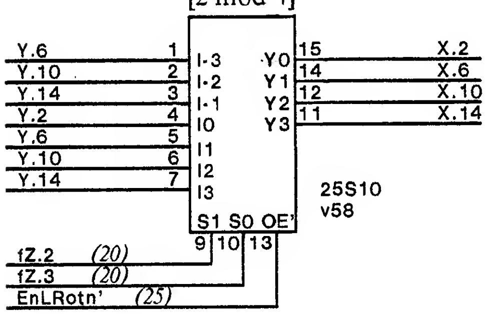
[0 mod 4]



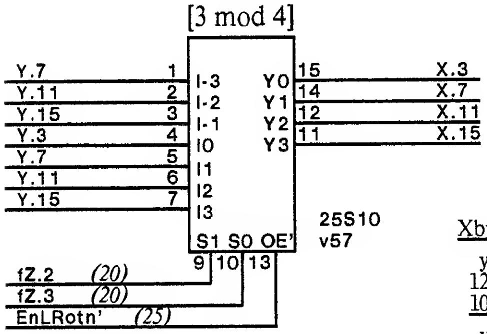
[1 mod 4]



[2 mod 4]



[3 mod 4]



fZ.2	fZ.3	
0	0	Left 0
0	1	Left 12
1	0	Left 8
1	1	Left 4

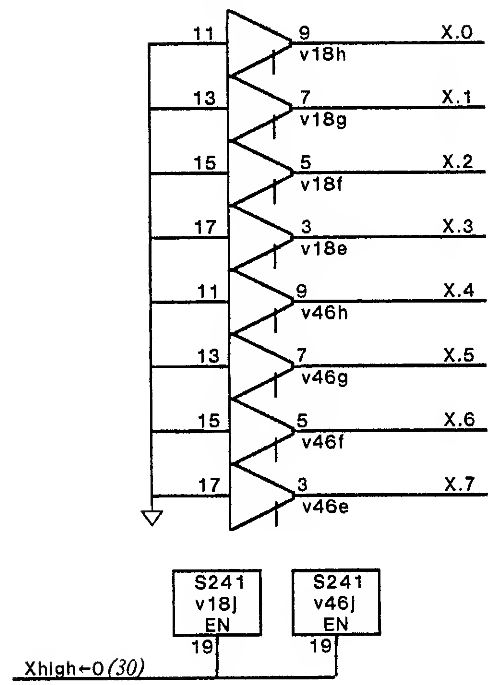
Zero disable X-bus

30	↑ to Xhigh←0
15	S241 EN to X-bus
10	X-bus
55 nS	

Xbus[0-7] ← 0

30	↑ to Xhigh←0
15	S241 OE
10	X-bus
55 nS	

Zero High XBus



Xbus ← Y LRotn = max (y+22, 56, 50) nS

y	↑ to Y bus
12	25S10 data in to out
10	X-bus
y +	22 nS

25	↑ to EnLRotn'
21	25S10 OE
10	X-bus
56 nS	

20	↑ to fZ.2
20	25S10 Select to X-bus
10	X-bus
50 nS	

LRotn disable X-bus

25	↑ to EnLRotn'
15	25S10 OE' to X-bus
10	X-bus
50 nS	

IB disable X-bus

25	↑ to XBus←IB'
14	S257 E' to X-bus
10	X-bus
49 nS	

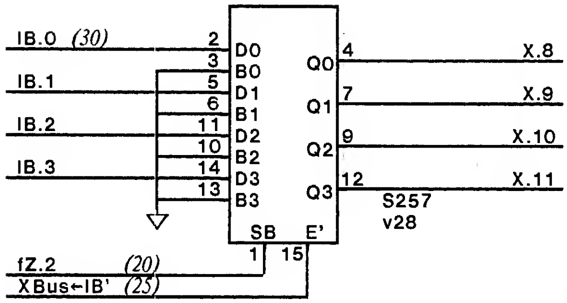
Byte disable X-bus

25	↑ to Nibble'
14	S257 E' to X-bus
10	X-bus
49 nS	

Nibble disable X-bus

25	↑ to Nibble'
15	S241 EN' to X-bus
10	X-bus
50 nS	

0, ibHigh



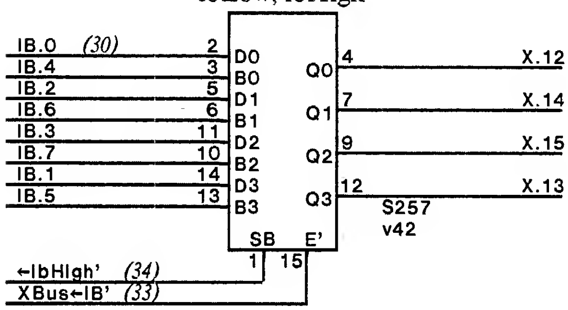
Xbus←IB = max(56,56,59) nS

34[4]	↑ to IB
8	S257 data to Xbus
10	X-bus
52[4] = 56 nS	

25	↑ to Xbus←IB'
21	S257 E' to Xbus
10	X-bus
56 nS	

34	↑ to←ibHigh'
15	S257 SB to Xbus
10	X-bus
59 nS	

ibLow, ibHigh



Xbus ← Nibble = max(39, 50) nS

20	↑ to fZ
9	S241 data to X-bus
10	X-bus
39 nS	

25	↑ to Nibble'
15	S241 EN' to X-bus
10	X-bus
50 nS	

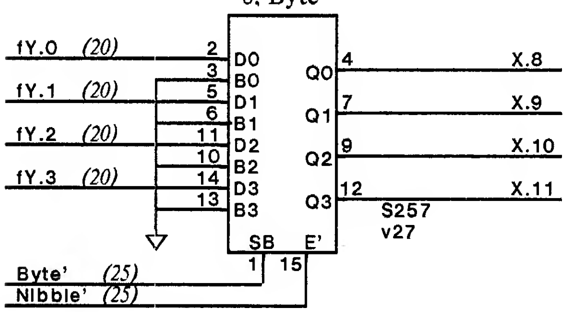
Xbus ← Byte = max(38, 56,50) nS

20	↑ to fY
8	S257 data to X-bus
10	X-bus
38 nS	

25	↑ to Nibble'
21	S257 E' to X-bus
10	X-bus
56 nS	

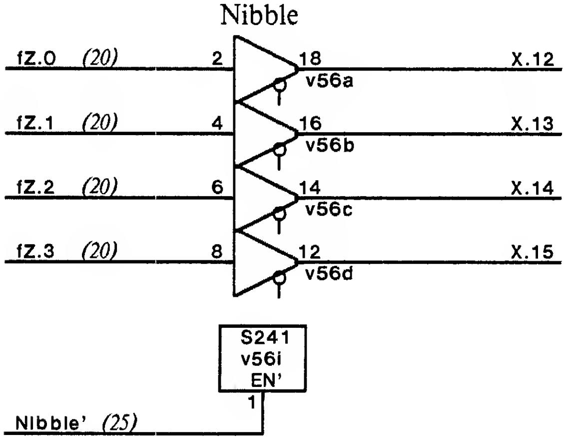
25	↑ to Byte'
15	S257 SB to Xbus
10	X-bus
50 nS	

0, Byte

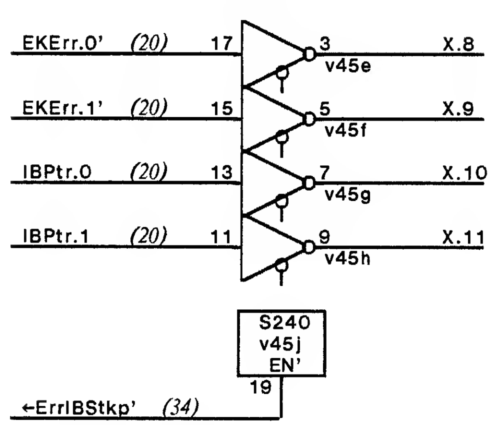


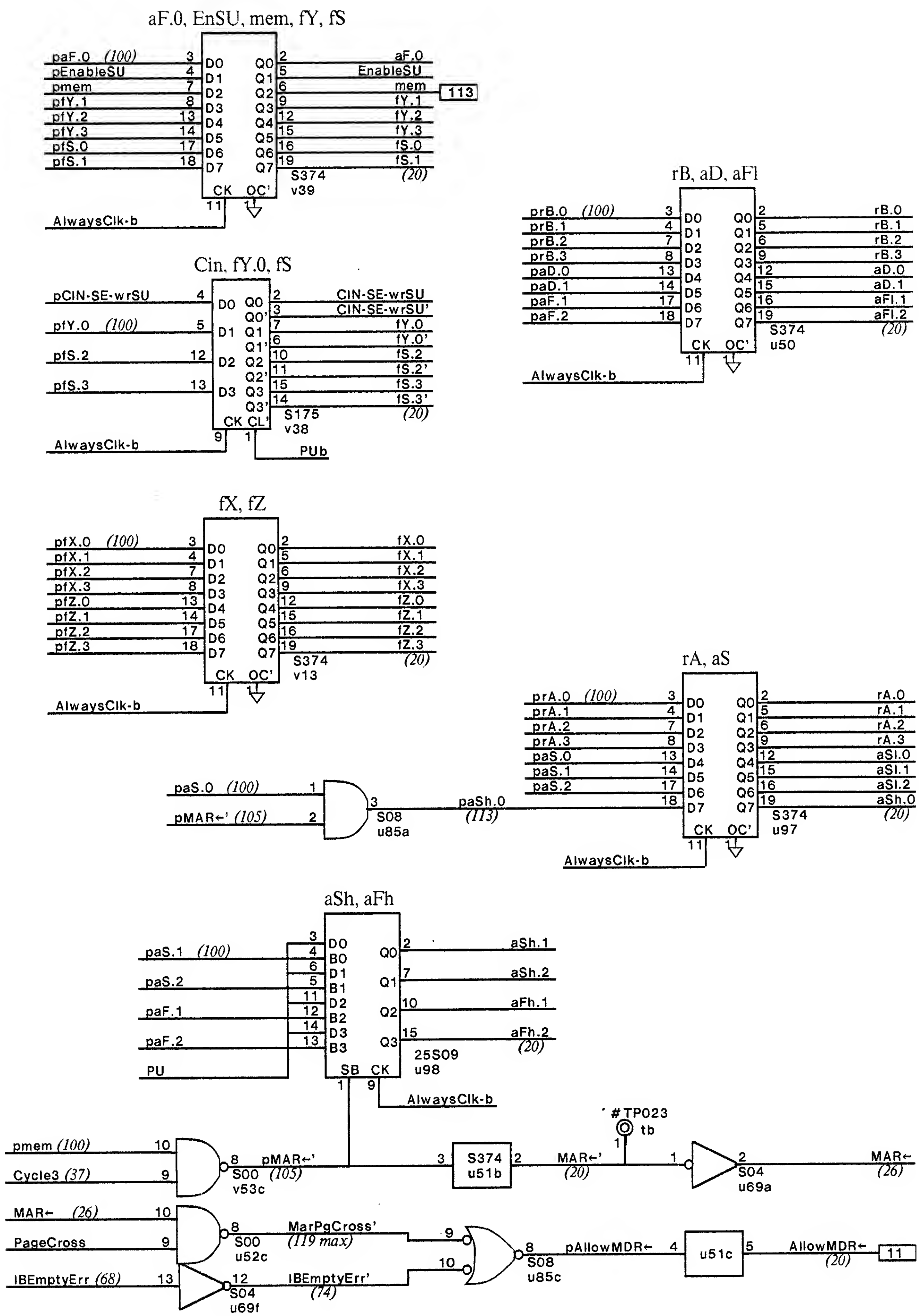
See stackP timings for ErrIBPtr

Nibble



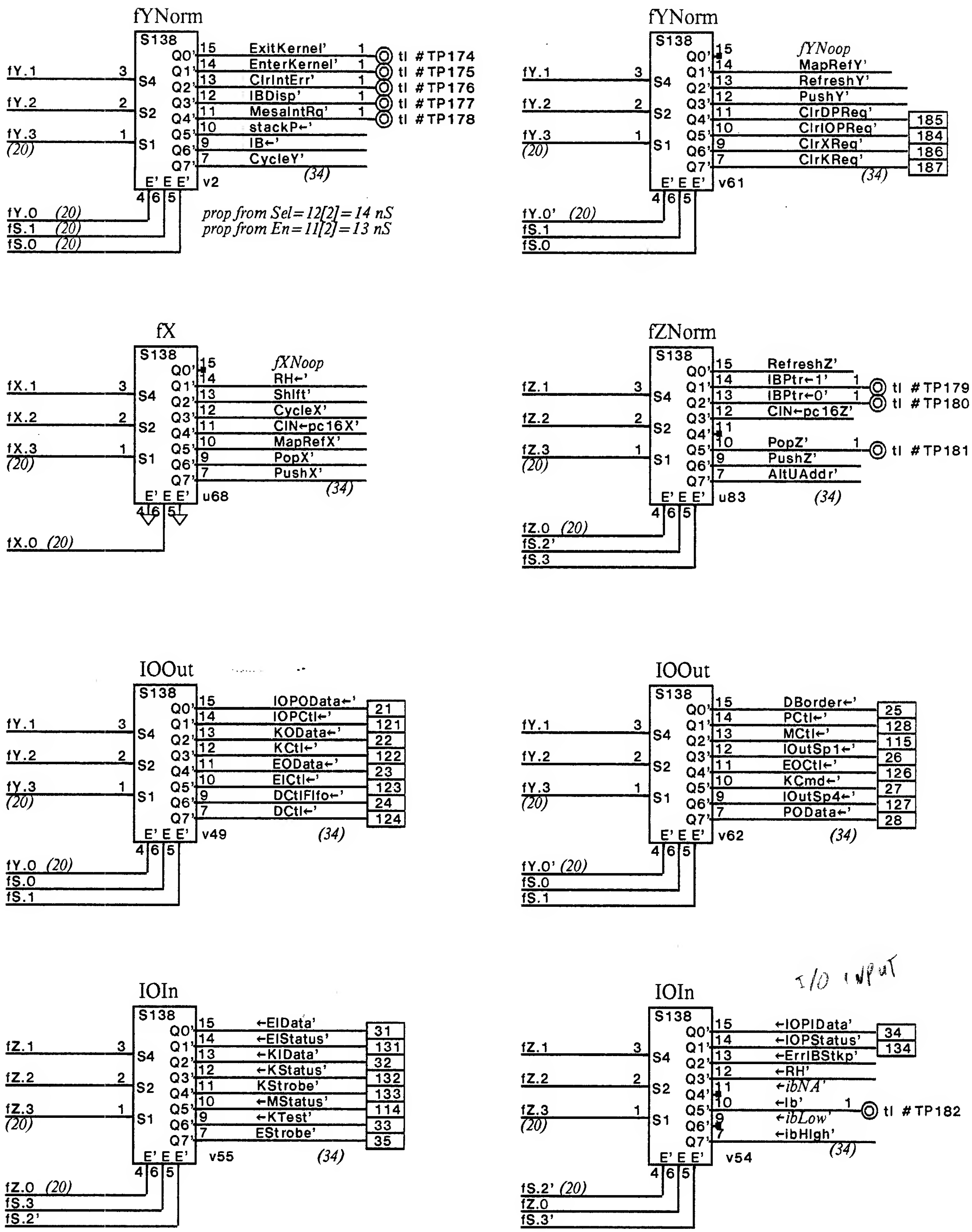
ErrIBPtr



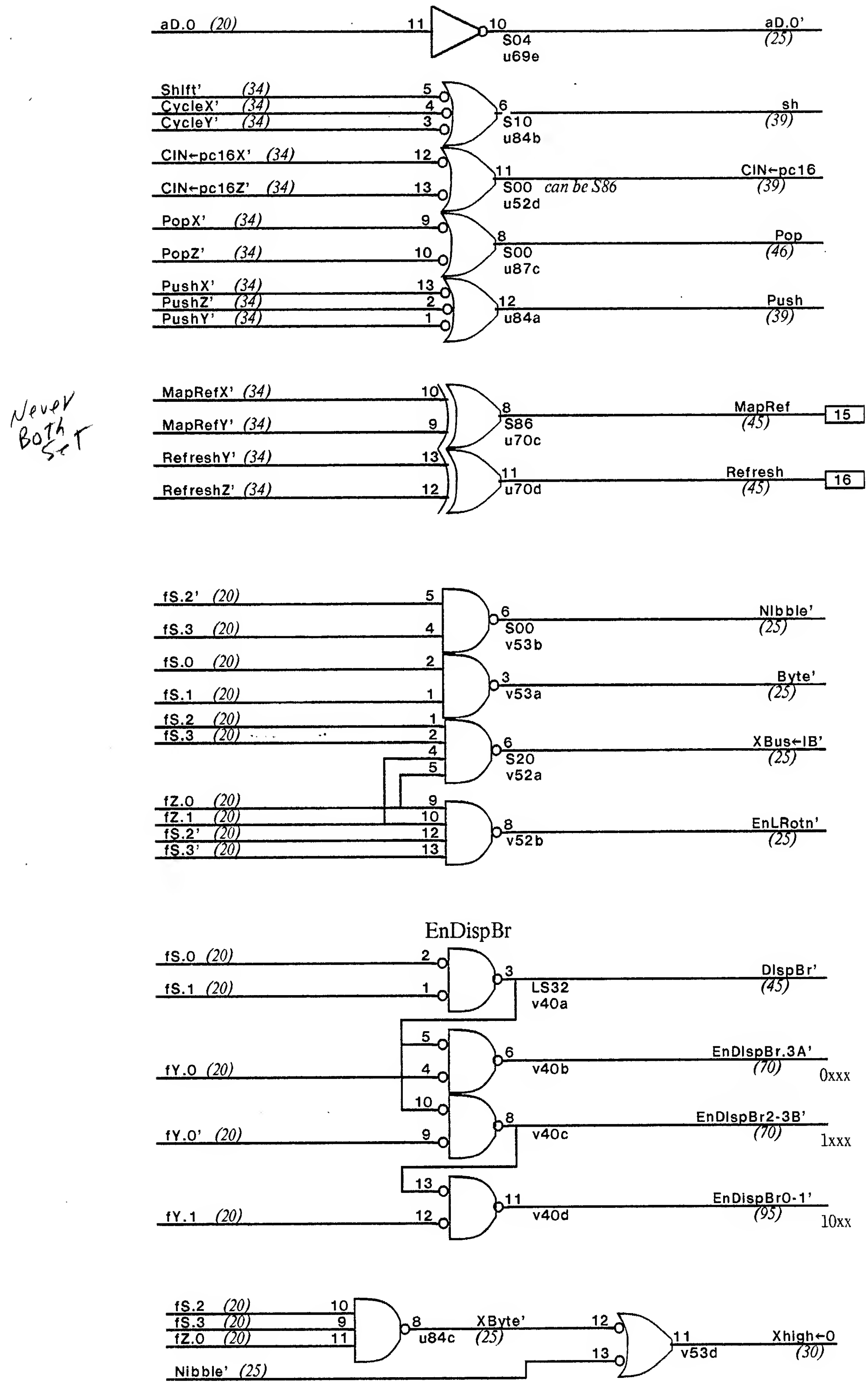


XEROX SDD	Project Dandelion	MIR	File pLionHead08.sil	Designer Garner	Rev M	Date 8/24/80	Page 08

Function Field Decoders



XEROX SDD	Project Dandelion	MIR Decoding I	File pLionHead09.sil	Designer Garner	Rev M	Date 12/4/80	Page 09



XEROX SDD	Project Dandelion	MIR Decoding II	File pLionHead10.sil	Designer Garner	Rev M	Date 4/3/…	Page 10

DispBr[0-1] = max(c+32,69,133)

20	↑ to fY
24[3]	S151 select to DispBr
18	DispBr' setup
64[3]=69	

95	↑ to EnDispBr0-1'
18[2]	S151 E' to DispBr
18	DispBr' setup
131[2] = 133 nS	

c	condition source
12[2]	S151 data to DispBr
18	DispBr' setup
c+30[2]= c+32	

DsipBr setup

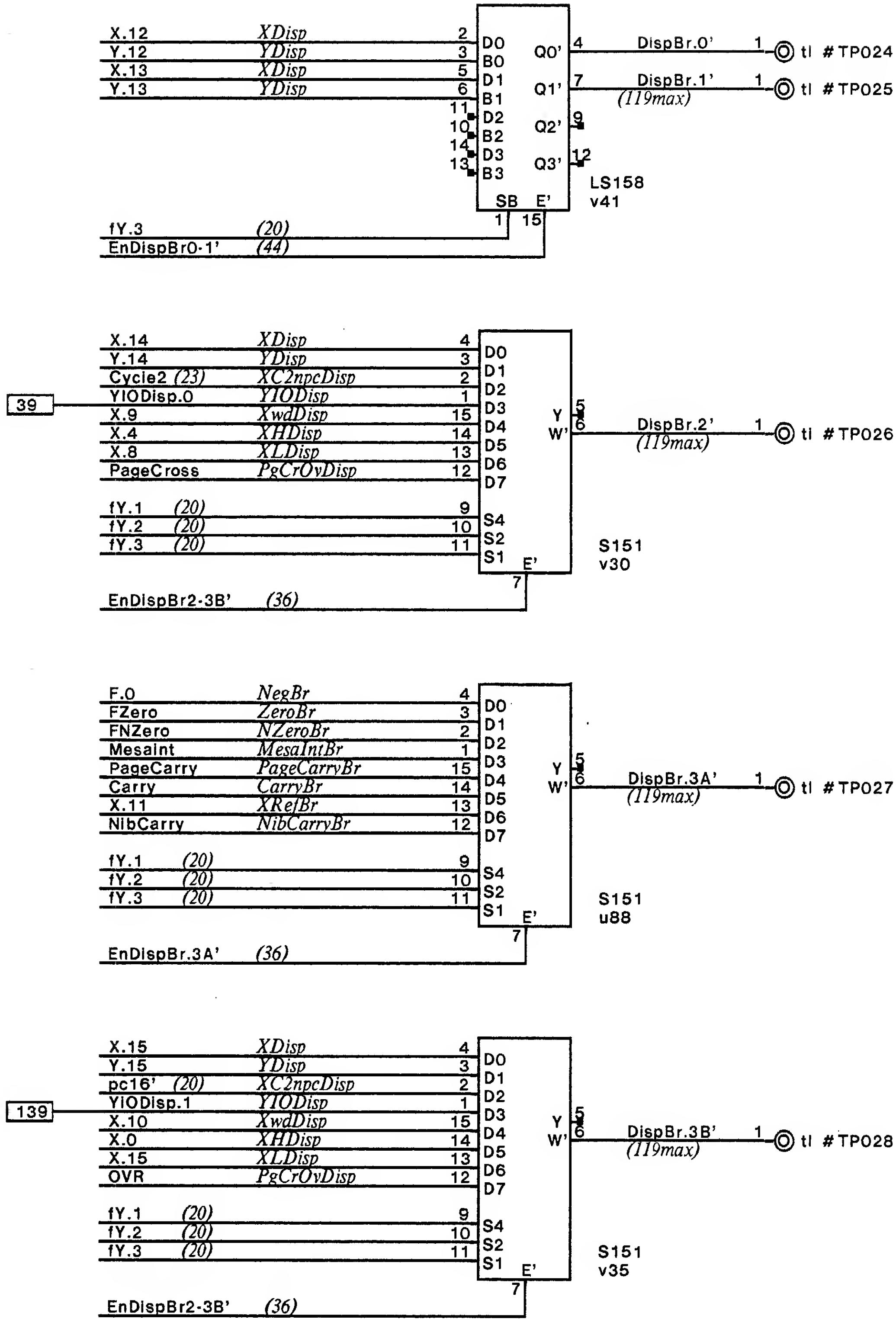
5	S00 in to pTC
6[1]	S64 in to pNIA
5[1]	25S09/S374 setup
18 nS	

DispBr[2-3]=max(c+26,55,103)

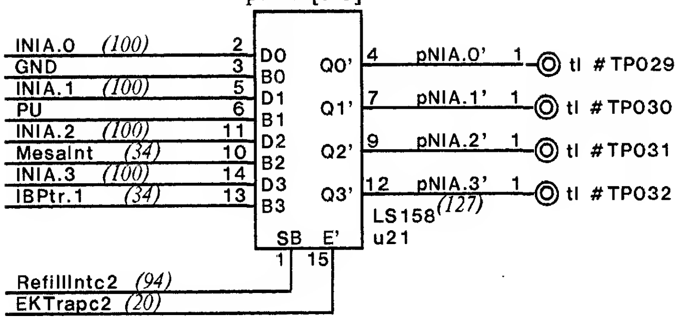
20	↑ to fY
15[2]	S151 select to DispBr
18	DispBr' setup
51[4]=55 nS	

70	↑ to EnDispBr.3A'
13[2]	S151 E' to DispBr
18	DispBr' setup
101[2] = 103 nS	

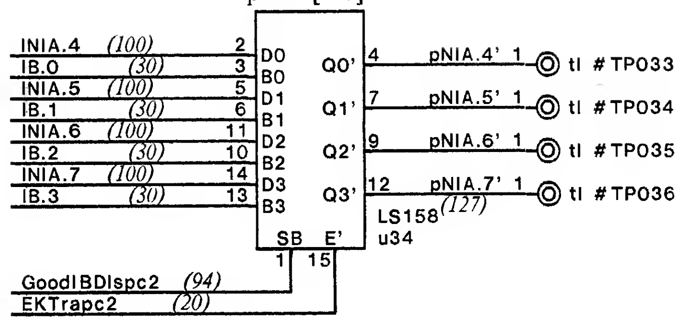
c	condition source
7[1]	S151 data to DispBr
18	DispBr' setup
c+23[3]=c+26 nS	



pNIA[0-3]



pNIA[4-7]

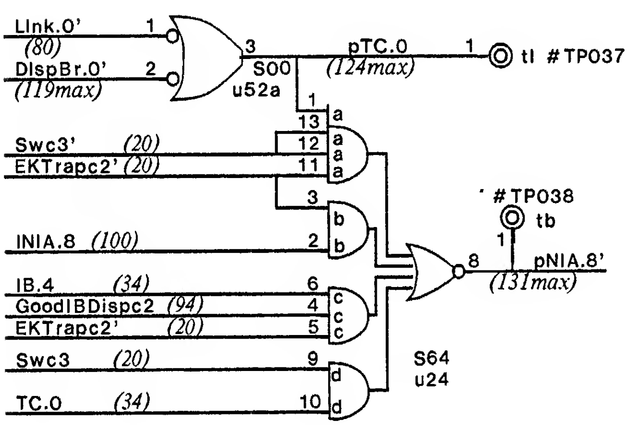


pNIA[0-7] = max(127, 120, 46) nS

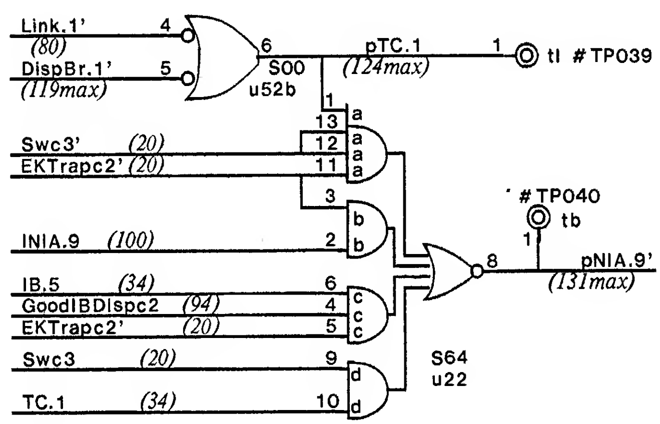
94	↑ to RefillIntc2	100	↑ to INIA	20	↑ to EKErrc2
24[3]	LS158 SB to pNIA'	12[2]	LS158 data to pNIA'	18[2]	LS158 E' to pNIA'
5[1]	25S09/S374 setup	5[1]	25S09/S374 setup	5[1]	25S09/S374 setup
123[4] = 127 nS		117[3] = 120 nS		43[3] = 46nS	

(See page 11 for pNIA[8-11] timing)

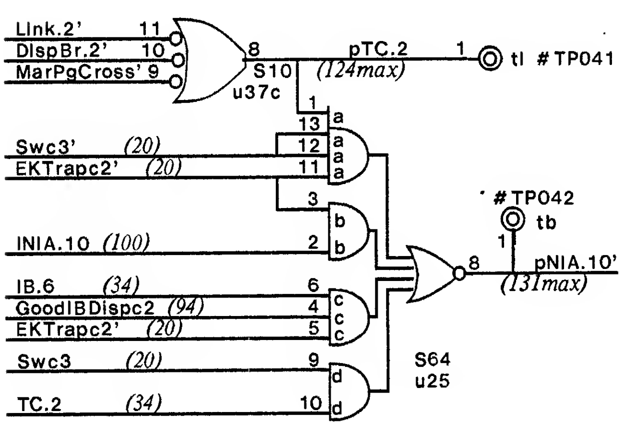
pNIA.8



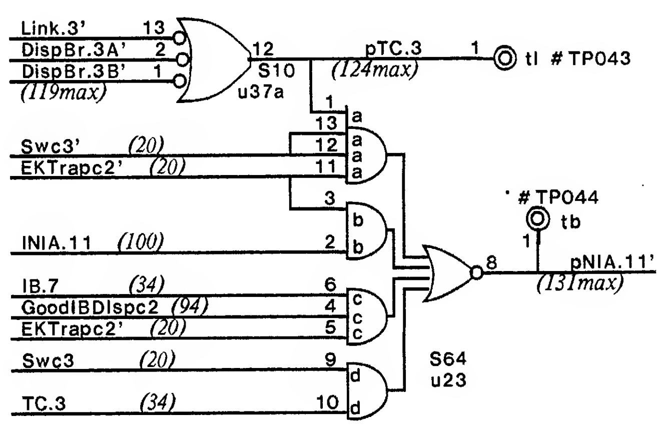
pNIA.9

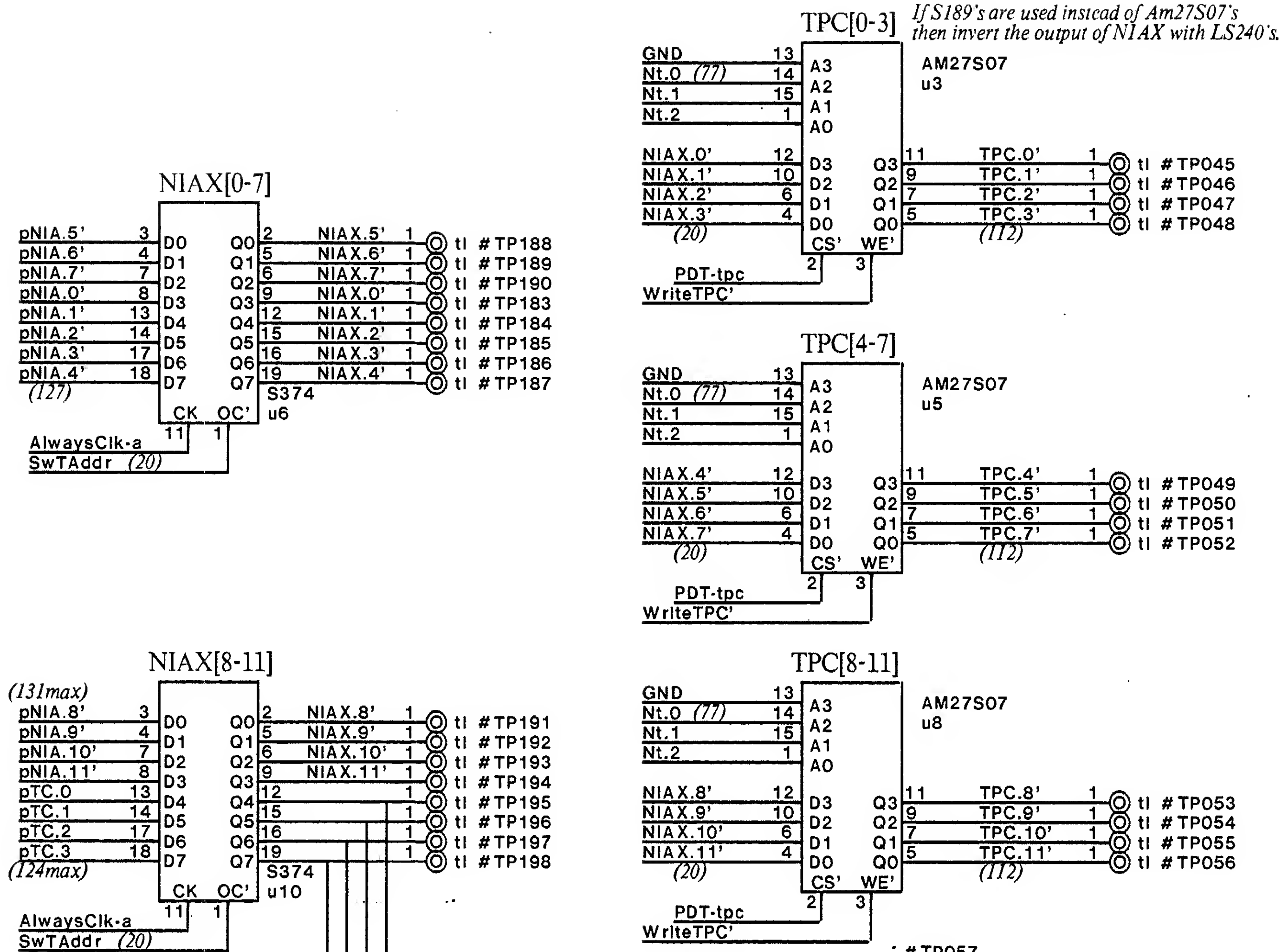


pNIA.10



pNIA.11





TPC/TC timing

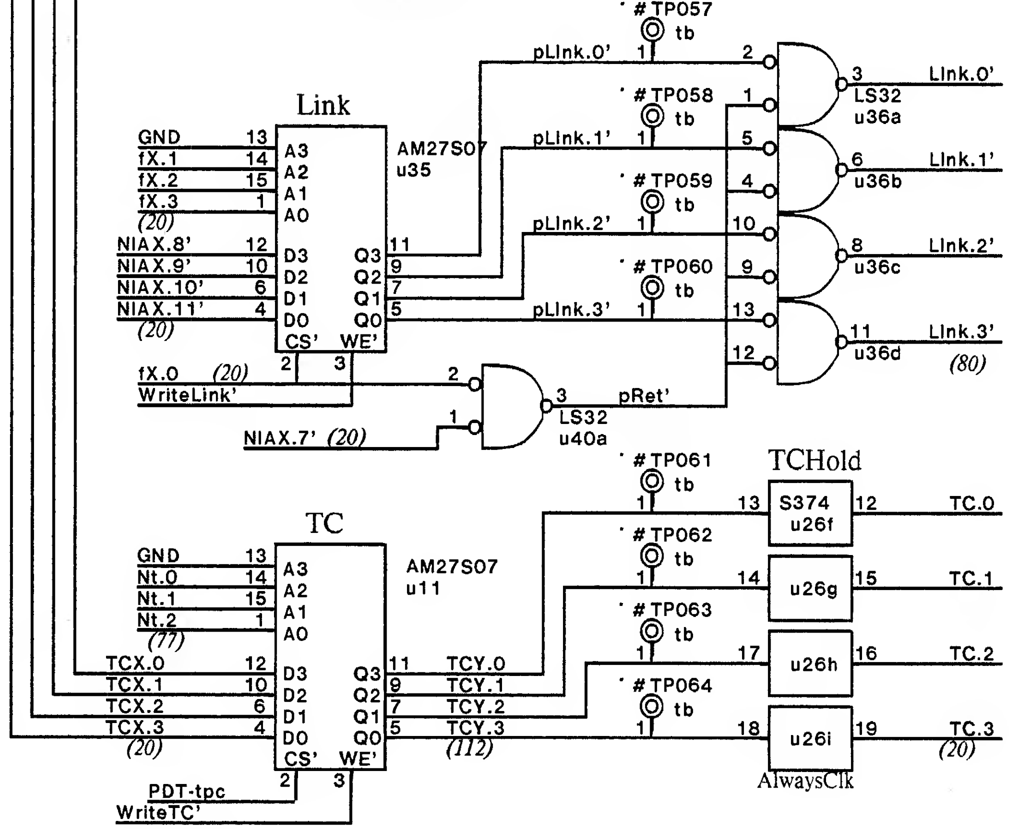
77	↑ to Nt
35	Am27S07 tAA
5[1]	25S09/S374 setup
117[1] = 118 nS	

Link timing

20	↑ to fX
35	Am27S07 tAA
22[3]	pLink' to Link'
18	DispBr' setup
95[3] = 98 nS	
20	↑ to fX.0, NIAX.7'
22[3]	fX.0 to pRet'
22[3]	pRet' to Link'
18	DispBr' setup
82[6] = 88 nS	

If only pullups were used on output of Link (instead of the LS32 kludge), then Link timing would be:

98	*WriteLink' active*
25[3]	*WE' to pLink high*
18	*DispBr' setup*
141[3] = 144 nS	



XEROX SDD	Project Dandelion	TPC, TC, & Link	File pLionHead13.sil	Designer Garner	Rev M	Date 12/4/80	Page 13

I/O Requests

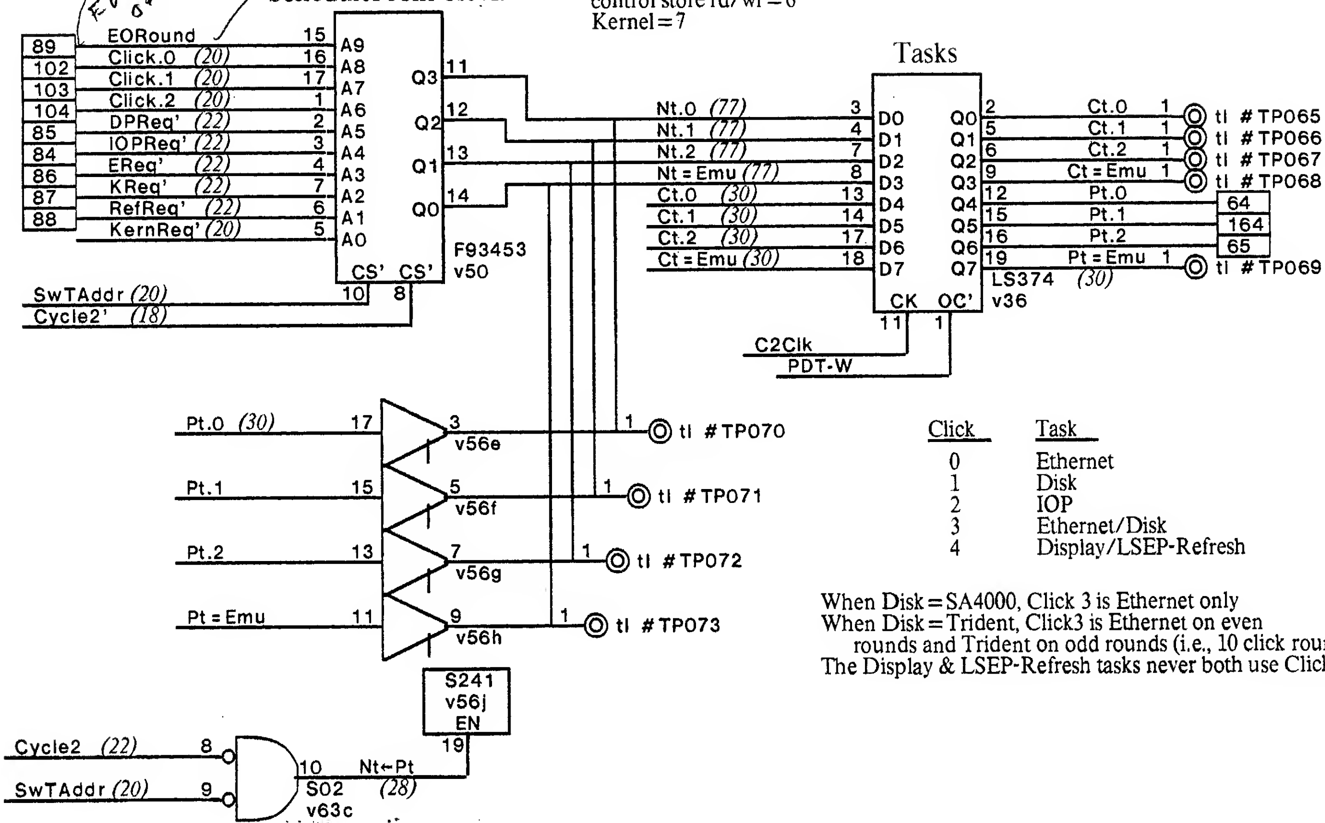
Even odd

Alternatis for Trident Disk Controller norm machine is ground

K=Disk

Nt, Ct, Pt

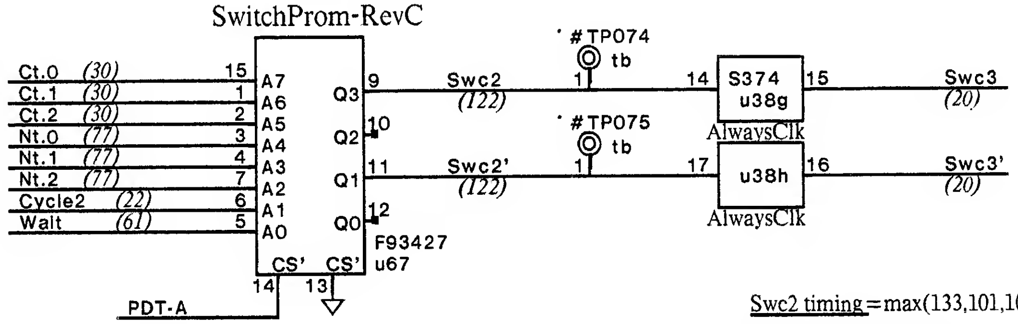
Emulator=0
DisplayLSEP=1
Ethernet=2
Refresh=3
Disk=4
IOP=5
control store rd/wr=6
Kernel=7



Click	Task
0	Ethernet
1	Disk
2	IOP
3	Ethernet/Disk
4	Display/LSEP-Refresh

When Disk=SA4000, Click 3 is Ethernet only
When Disk=Trident, Click3 is Ethernet on even rounds and Trident on odd rounds (i.e., 10 click round).
The Display & LSEP-Refresh tasks never both use Click 4.

when To switch tasks

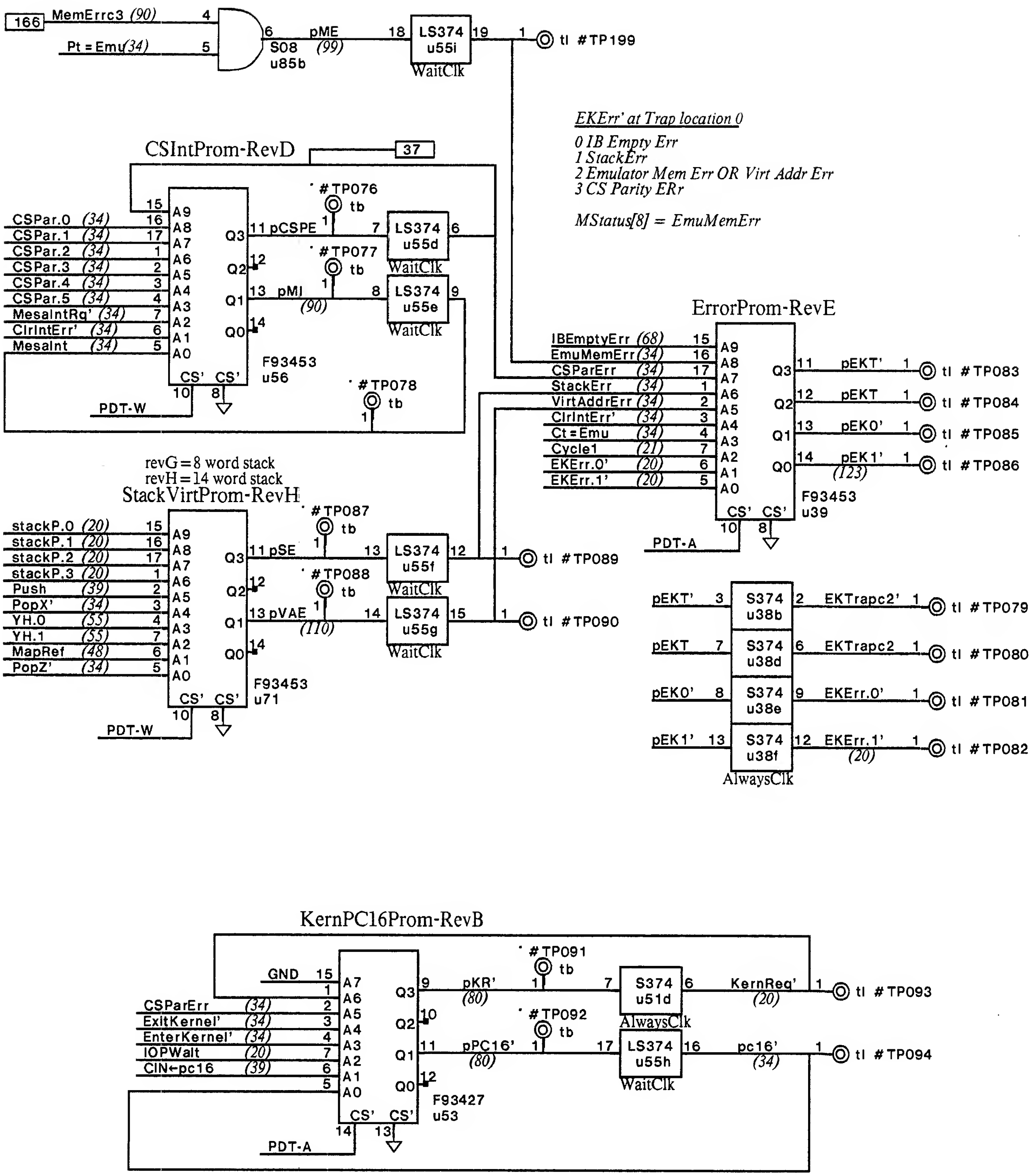


Swc2 timing=max(133,101,101)

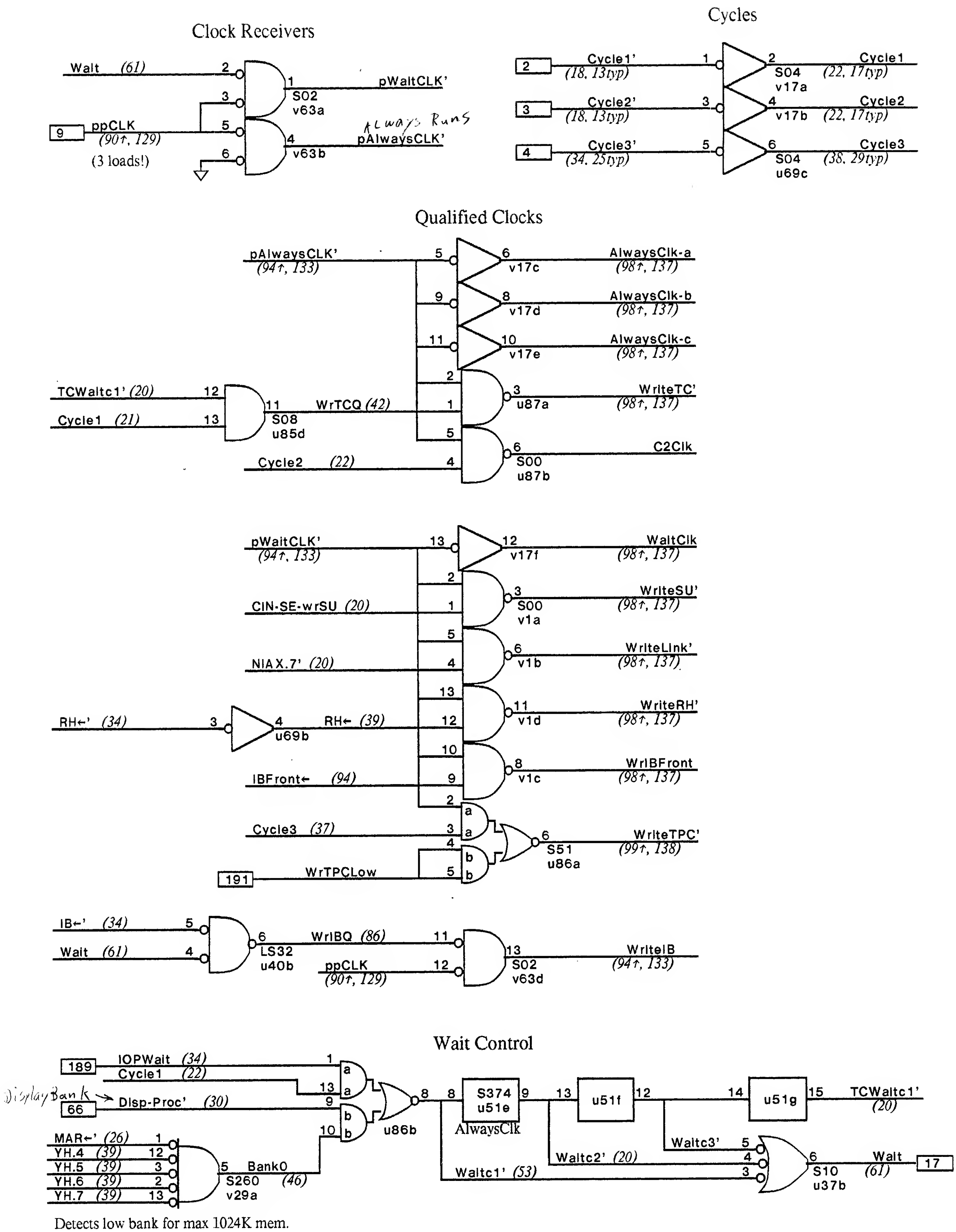
22	↑ to Kreq'
55	F93453 addr to Nt
45	F93427 addr to Swc2
10[1]	25S09 SB setup
132[1]=133 nS	
20	↑ to SwTAddr
25	F93453 CS' to Nt
45	F93427 addr to Swc2
10[1]	25S09 SB setup
100[1]=101 nS	
28	↑ to Nt←Pt
15[2]	S241 EN to Nt
45	F93427 addr to Swc2
10[1]	25S09 SB setup
98[3]=101 nS	

	Nt (Prom)	Nt	Ct	Pt
c1	3-S	Previous	Current	Previous
c2	Next	Next	Current	Previous
c3	3-S	Current	Next	Current

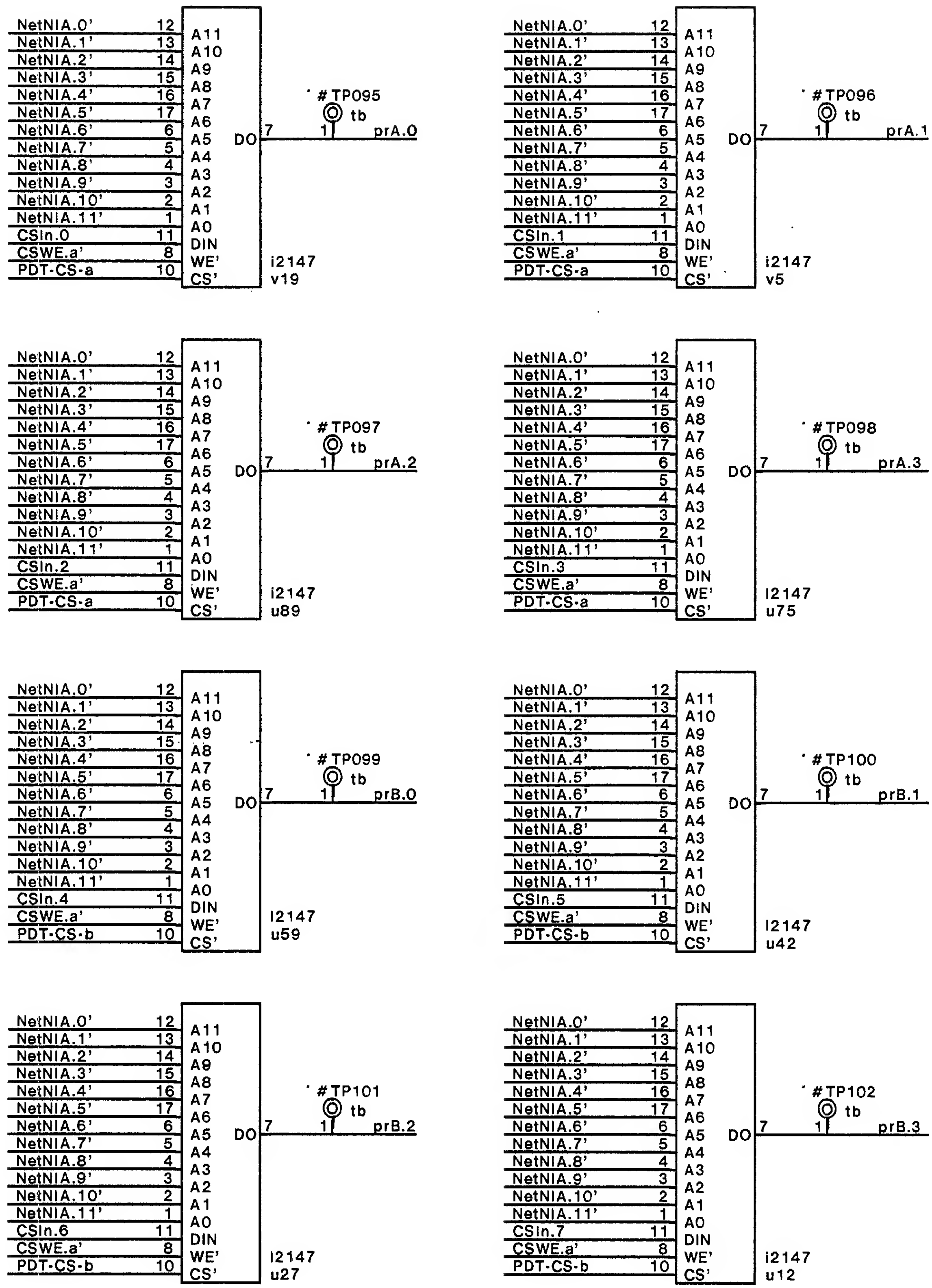
XEROX SDD	Project Dandelion	Schedule, Switch, & Tasks	File plionHead14.sil	Designer Garner	Rev M	Date 10/30/80	Page 14



XEROX SDD	Project Dandelion	Error, Emulator, & Kernel Proms	File pLionHead15.sil	Designer Garner	Rev M	Date 12/4/80	Page 15



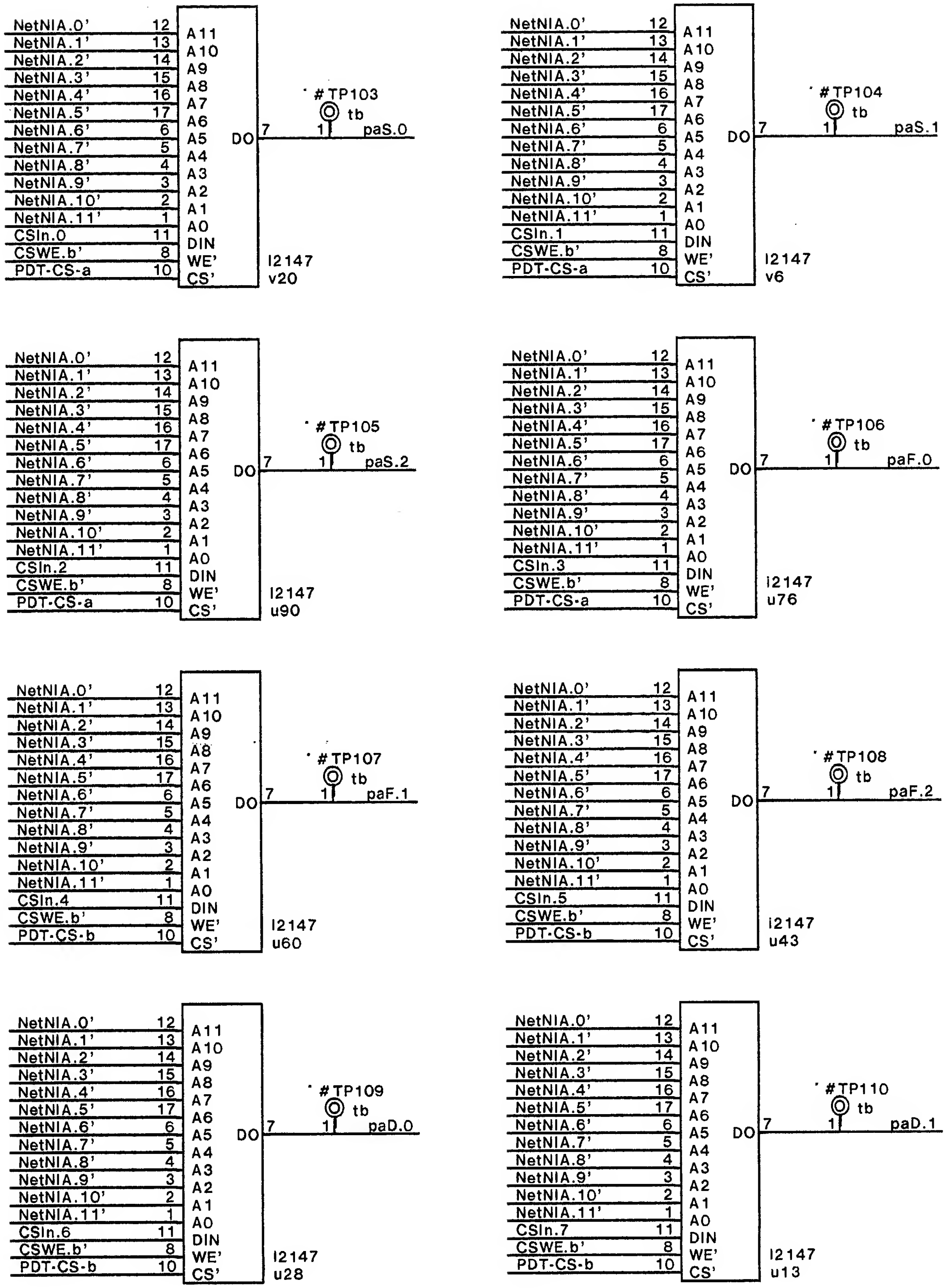
XEROX SDD	Project Dandelion	Clocks, Wait	File pLionHead16.sil	Designer Garner	Rev M	Date 8/23/80	Page 16



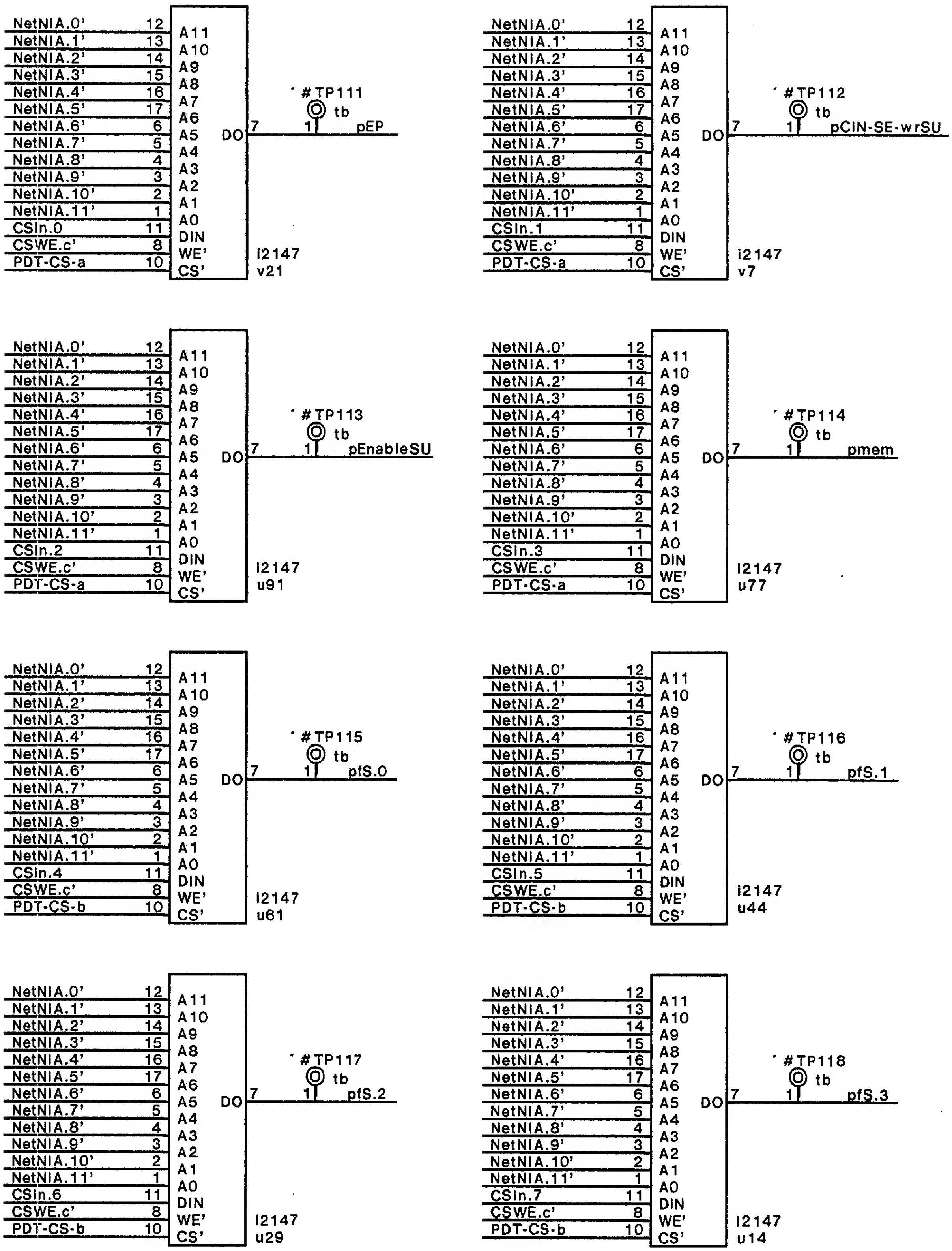
CS Timing

17 ↑ to NIA'
13 transmission delay
70 tAA 2147L
100 nS

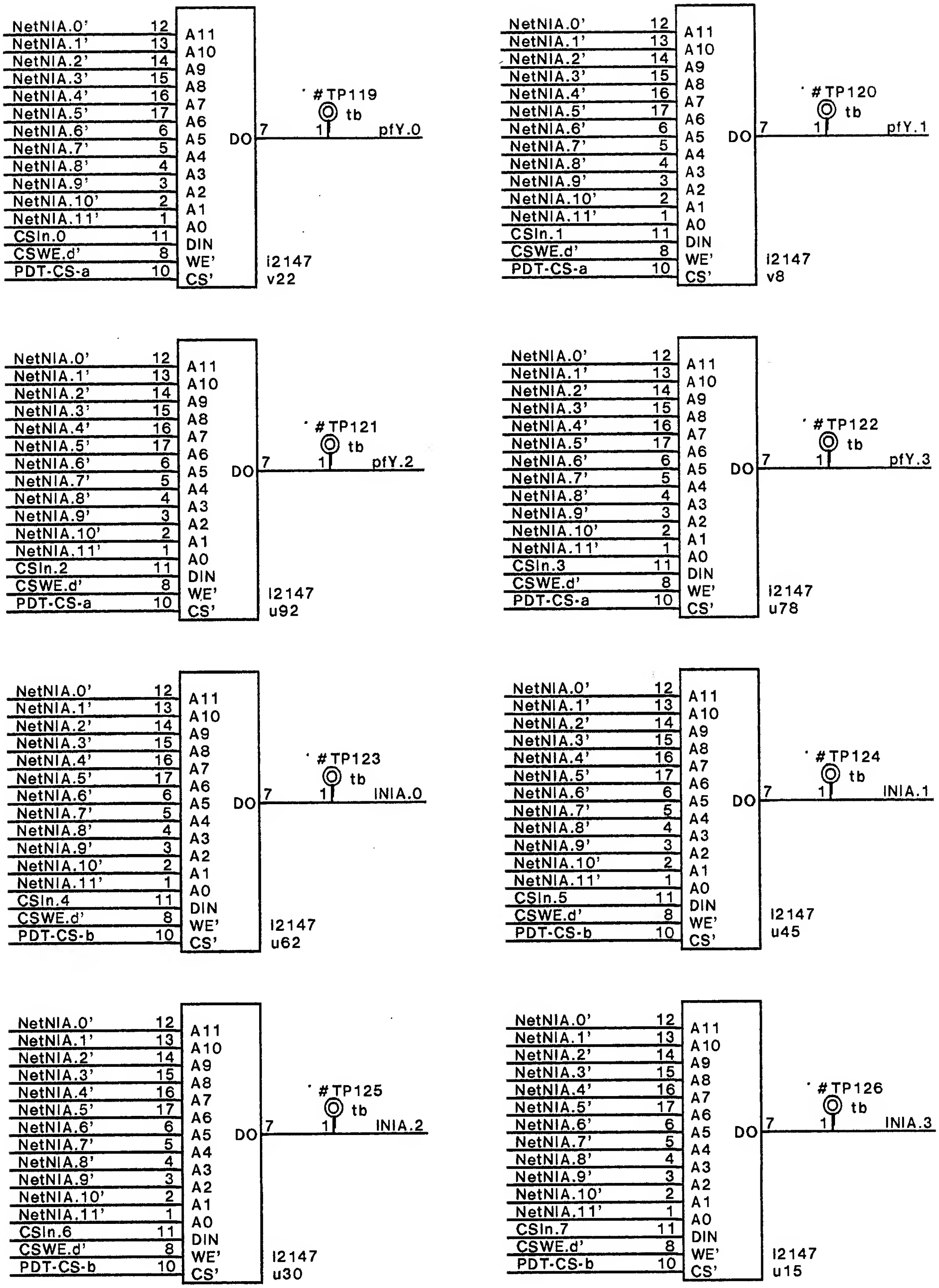
XEROX SDD	Project Dandelion	Control Store A [0-7]	File pLionHead17.sil	Designer Garner	Rev M	Date 5/14/80	Page 17



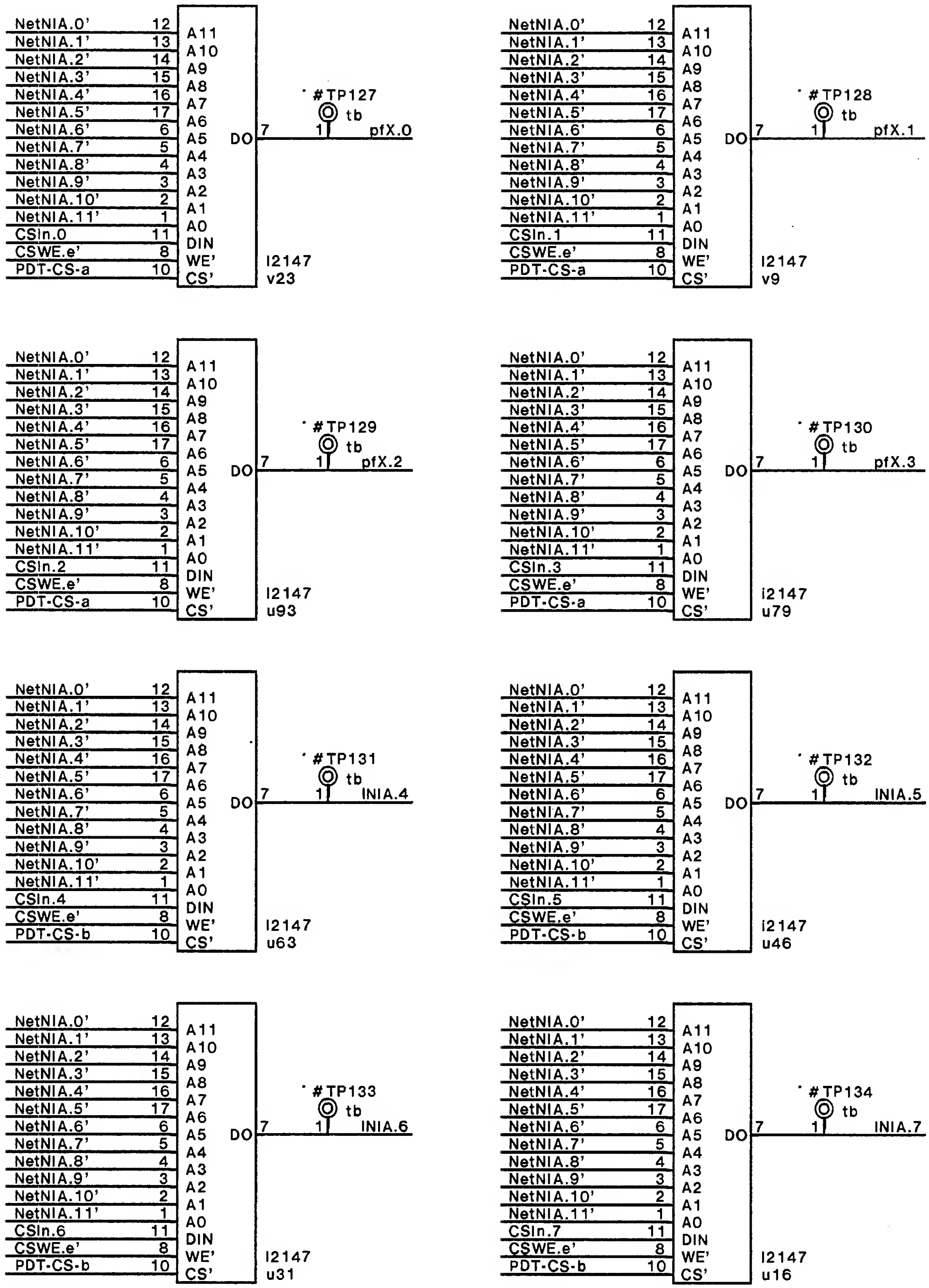
XEROX SDD	Project Dandelion	Control Store B [8-15]	File pLionHead18.sil	Designer Garner	Rev M	Date 5/14/80	Page 18



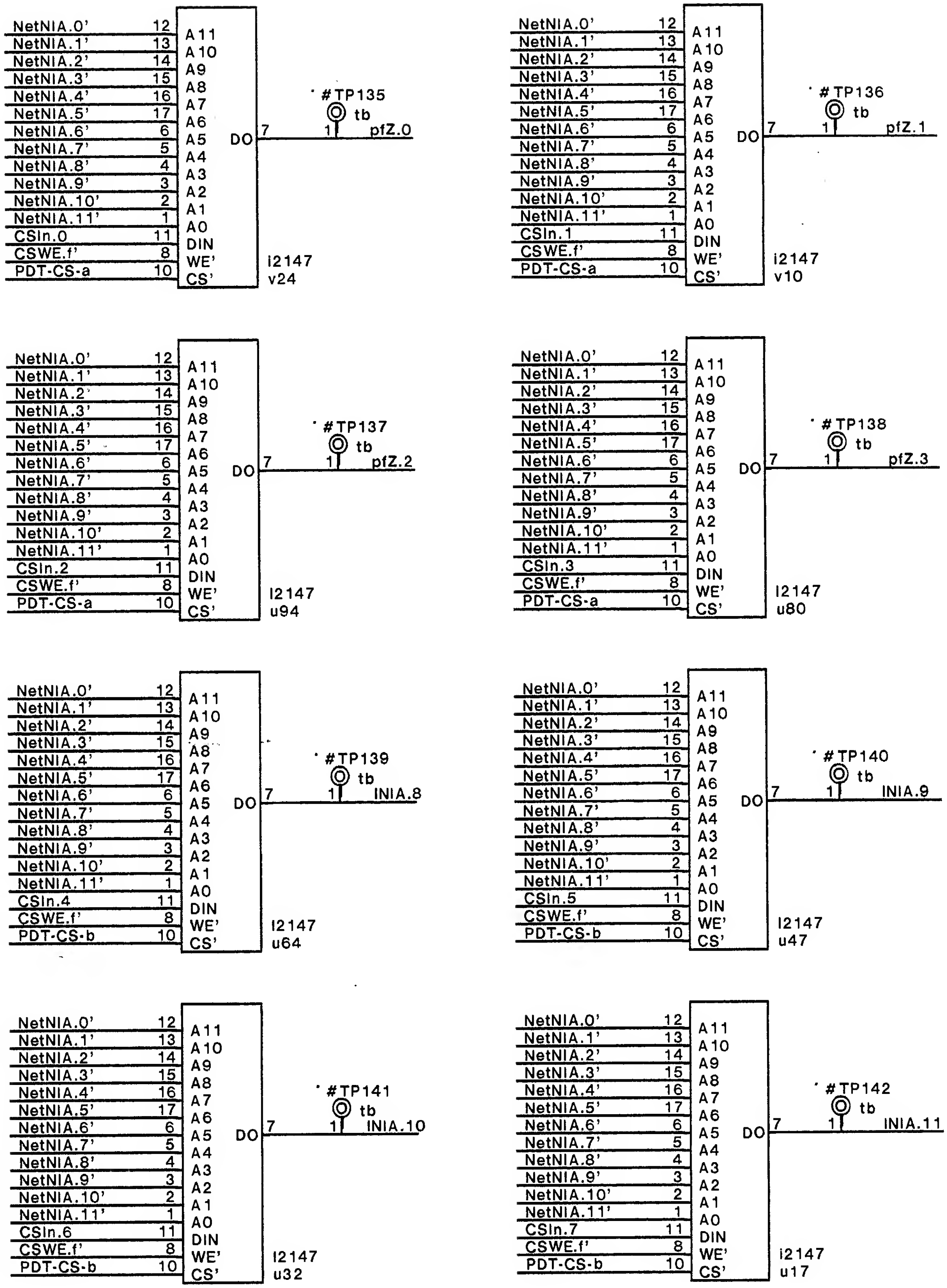
XEROX SDD	Project Dandelion	Control Store C [16-23]	File pLionHead19.sil	Designer Garner	Rev M	Date 5/14/80	Page 19



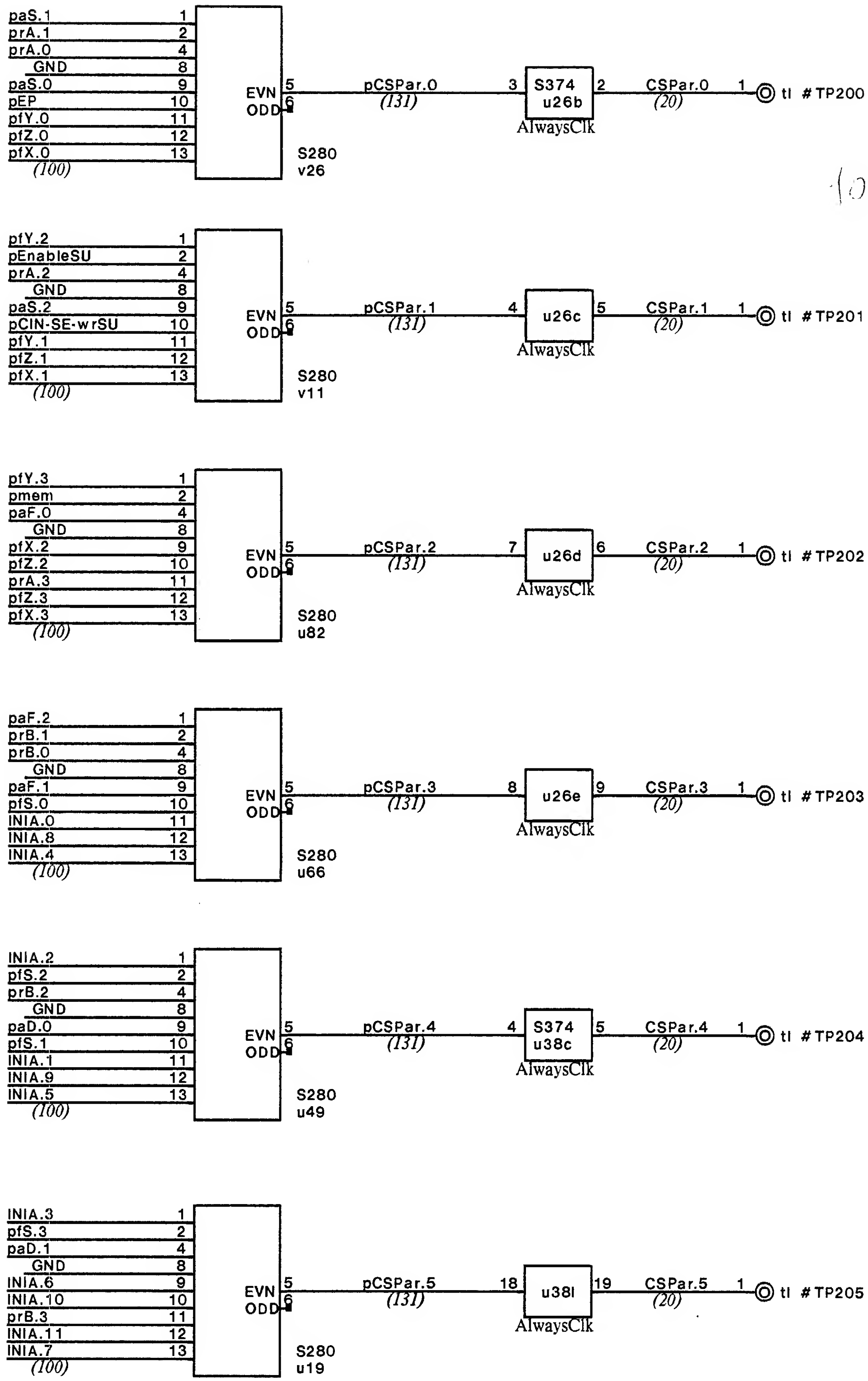
XEROX SDD	Project Dandelion	Control Store D [24-31]	File pLionHead20.sil	Designer Garner	Rev M	Date 5/14/80	Page 20



XEROX SDD	Project Dandelion	Control Store E [32-39]	File pLionHead21.sil	Designer Garner	Rev M	Date 5/14/80	Page 21



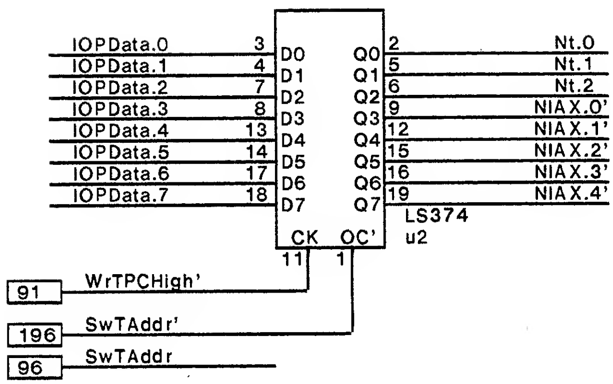
XEROX SDD	Project		File	Designer	Rev	Date	Page
	Dandelion	Control Store F [40-47]	pLionHead22.sil	Garner	M	8/23/80	22



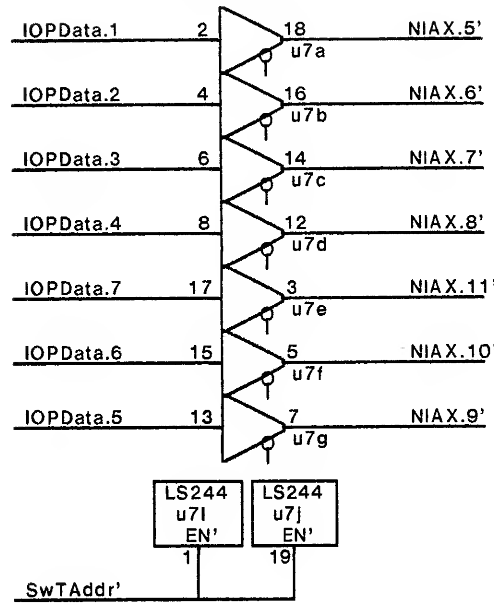
10 pg 15

XEROX SDD	Project Dandelion	CS Parity (PC) - Yek	File pLionHead23.sil	Designer Garner	Rev M	Date 12/4/80	Page 23

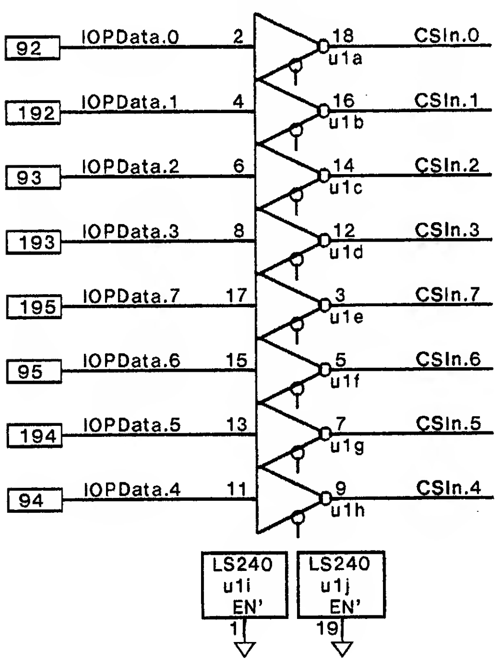
IOP TPCHigh



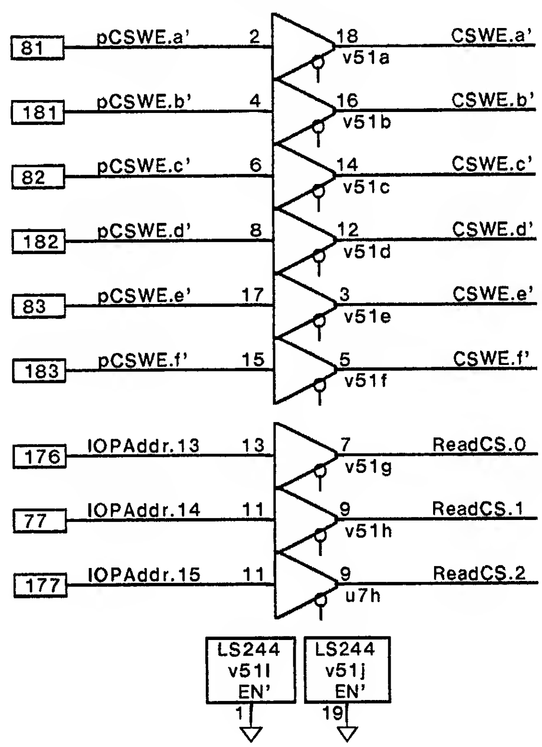
IOP TPCLow



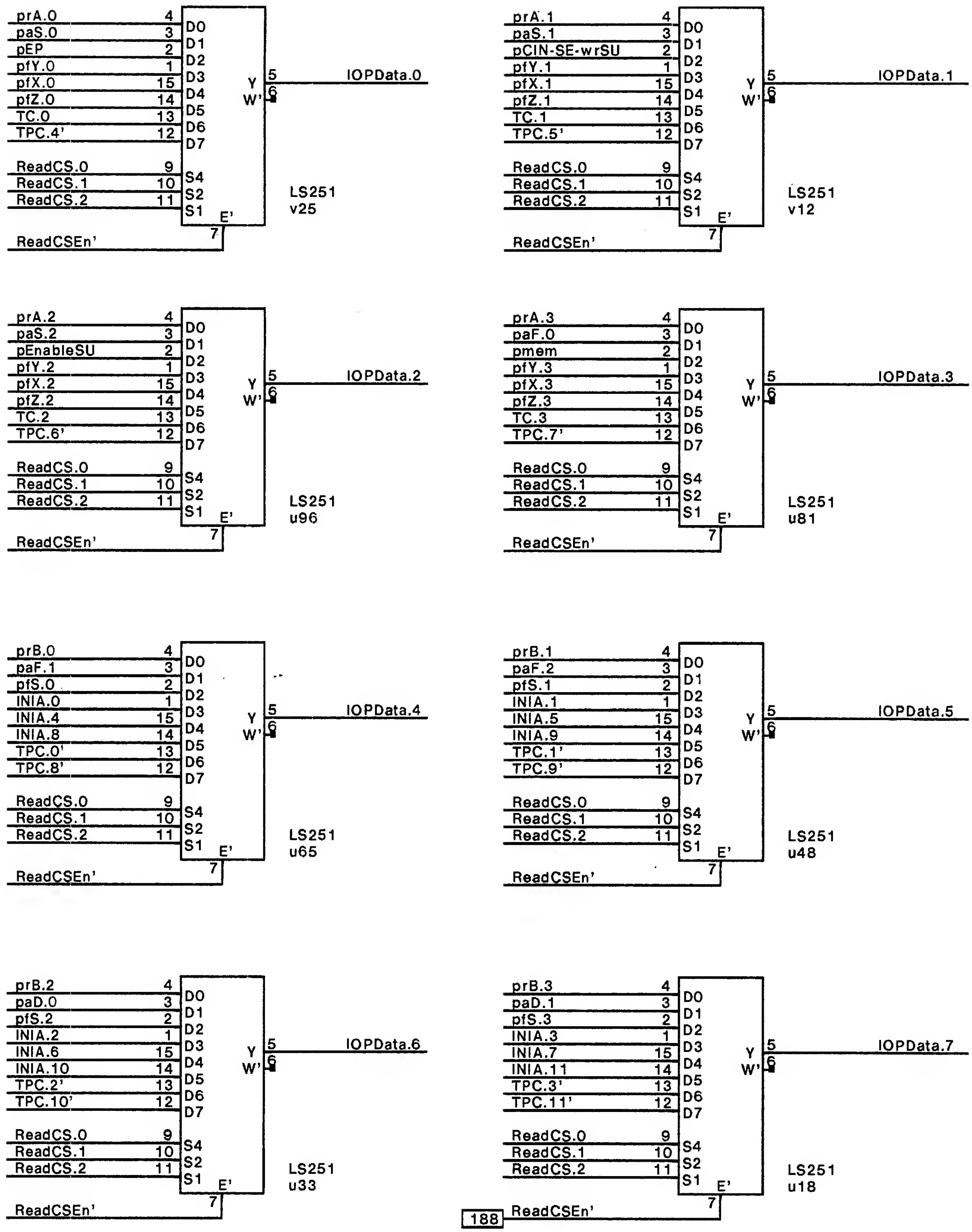
CS Data In



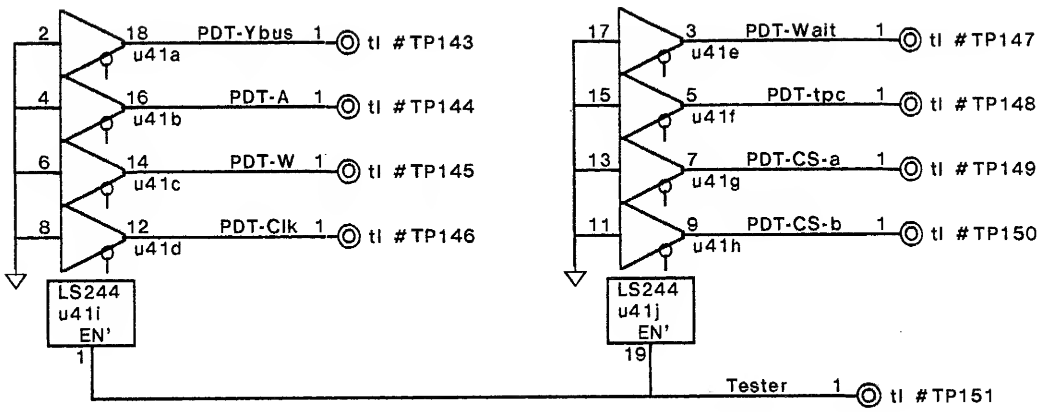
CSWE, IOPAddr Recv



XEROX SDD	Project Dandelion	Tasks. IOP TPC-TC Control	File pLionHead24.sil	Designer Garner	Rev M	Date 10/30/80	Page 24



XEROX SDD	Project		File	Designer	Rev	Date	Page
	Dandelion	CS Read	pLionHead25.sil	Garner	M	8/23/80	25



The Control Store can be read & written via backplane pins. Once tested, instructions (or parts of instructions), can be loaded in order to test additional features. For instance, all X-bus sources can be disabled by loading a 6 into CS bits 16-23 (controlled by CSWE.c'). Simple programs to test the 2901's can also be executed in this way.

The SU & RH registers can be loaded by controlling EnableSU, CIN-SE-wrSU, & RH← from a microinstruction.

stackP, IB, High SU Addr, & Low SU Addr can be similarly tested.

The MIR & MIR decoding can be tested by loading instructions into the CS.

PDT-Ybus is used to test devices attached to the Y bus.

PDT-A is used to disable registers or Proms whose outputs go to a register clocked by AlwaysClk.

PDT-W is similarly used for WaitClk.

PDT-Clk & PDT-Wait disable the outputs of AlwaysClk & WaitClk'd registers.

The following steps cause a CS byte to be written. It is assumed that the TPC has been written with the required CS address.

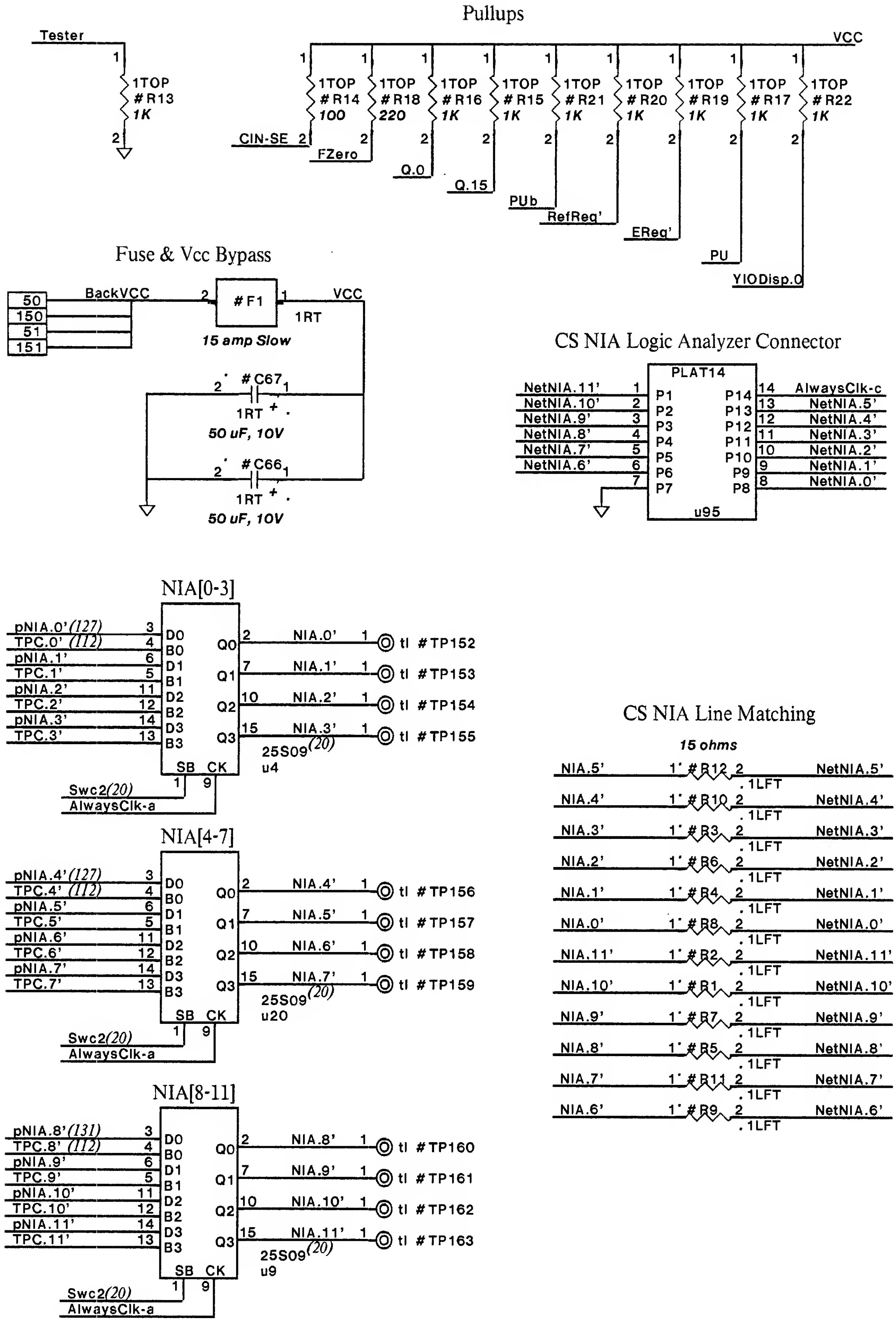
```
PDT-Clk ← 1; Swc3←1;              {cause NIA to come from TPC}
IOPWait ← 1;
SwTAddr' ← 0; SwTAddr ← 1;              {init code}
     IOPData ← data
     CSWE.x' ← 0; CSWE.x' ← 1;
```

If IOPWait is left high, the CP will not execute the instruction which has been loaded into the CS. Instead, the CP will be frozen in a state where the instruction is totally decoded, but the result will no be loaded into any register. Thus, all the microinstruction register (MIR) decoding logic can be tested without even executing an instruction.

The following steps cause the TPC to be written:

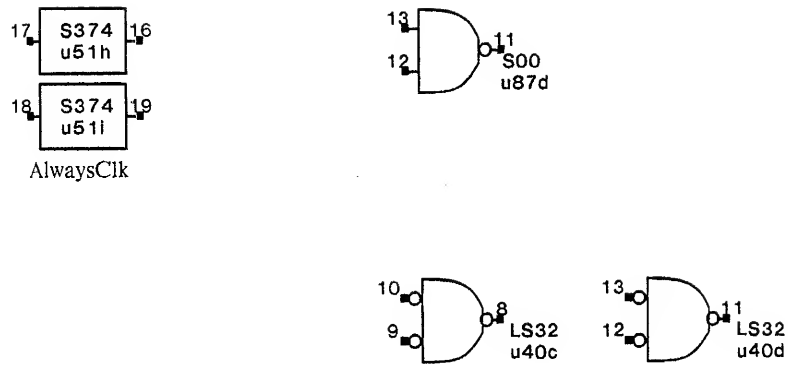
```
IOPWait ← 1;                                   {init code}
SwTAddr' ← 0; SwTAddr ← 1;
    IOPData ← (addr lshift5) or (data rshift7);          {set TPC addr & high 5 bits of data}
    WrTPCHigh' ← 0; WrTPCHigh' ← 1;
    IOPData ← data and 7F'x;                         {write low 7 bits}
    WrTPCLow ← 0; WrTPCLow ← 1;
```

D0 card test programs for reading & writting TPC & CS available on [Iris]<Workstation>LH>CardTest.dm



XEROX SDD	Project Dandelion	PC Discretes & NIA	File pLionHead27.sil	Designer Garner	Rev M	Date 10/30/80	Page 27

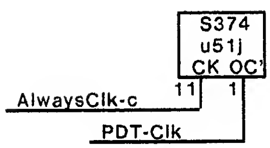
Unused Parts



Junk 374 Allocation

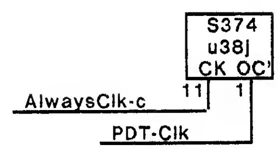
S374
g15 - Always Clock

b MAR←
c AllowMDR←
d KernReq'
e WaitC2
f WaitC3
g TCWait
h
i



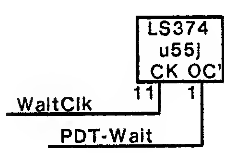
S374
h16 - Always Clock

b EKTrapc2'
c CSPar.4 *PC only*
d EKTrapc2
e EKErr.0'
f EKErr.1'
g Swc3
h Swc3'
i CSPar.5 *PC only*



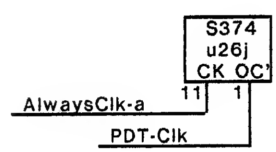
LS374
h17 - Wait Clock

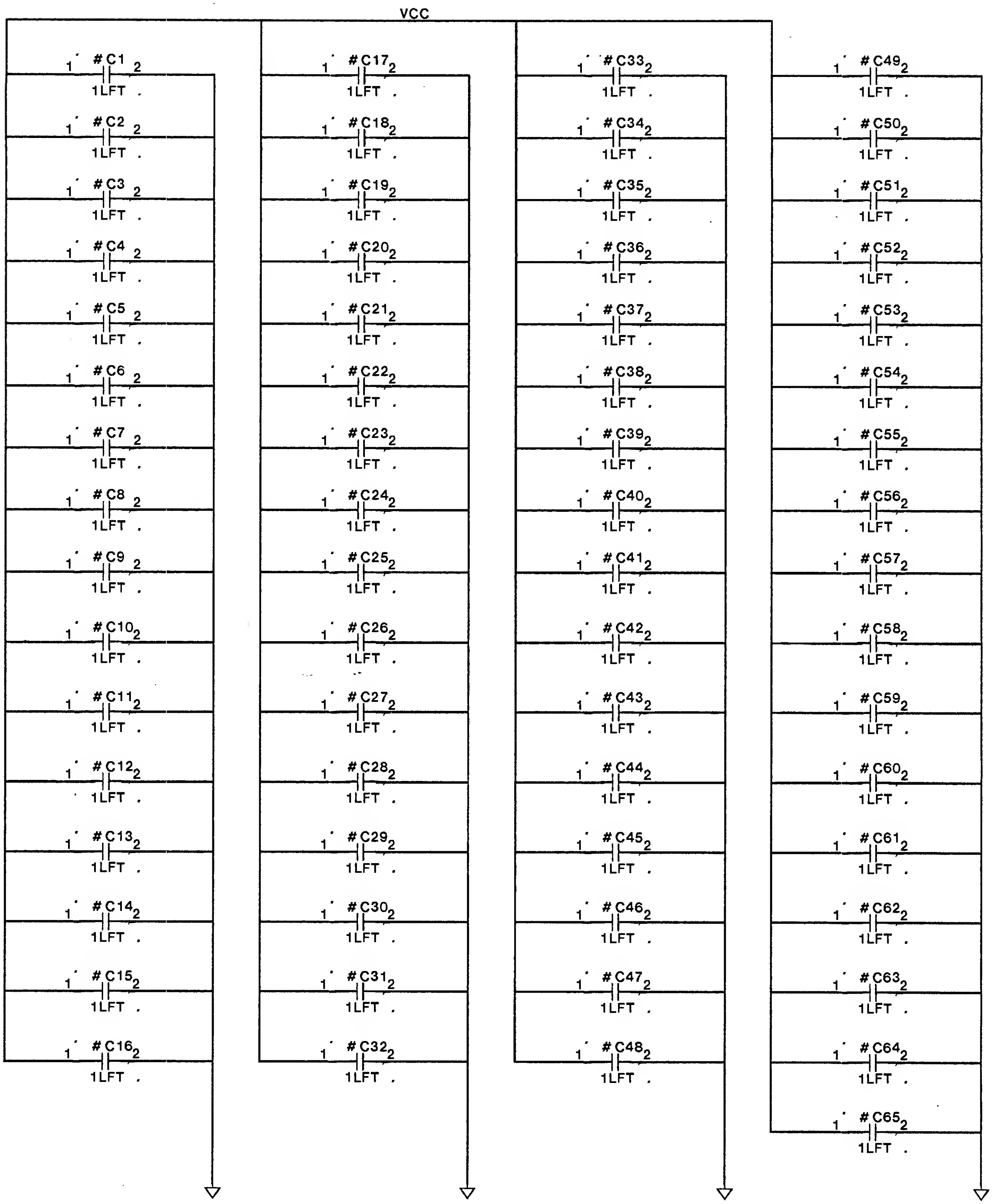
b IBPtr.0
c IBPtr.1
d CSParErr
e MesaInt
f StackErr
g VirtAddrErrc2
h pc16'
i MemErrc3



S374
f9 - Always Clock

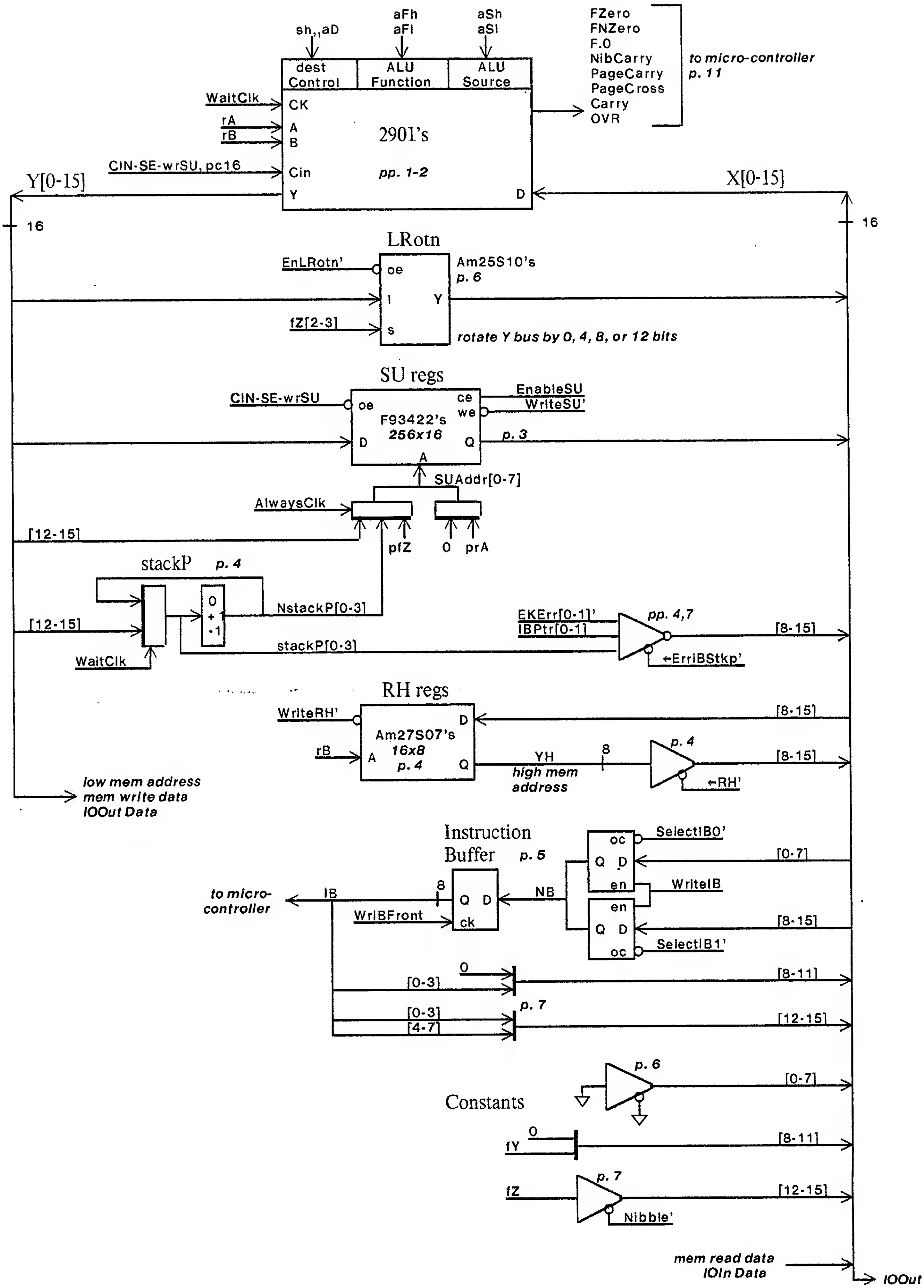
b CSPar.0
c CSPar.1
d CSPar.2
e CSPar.3
f TC.0
g TC.1
h TC.2
i TC.3



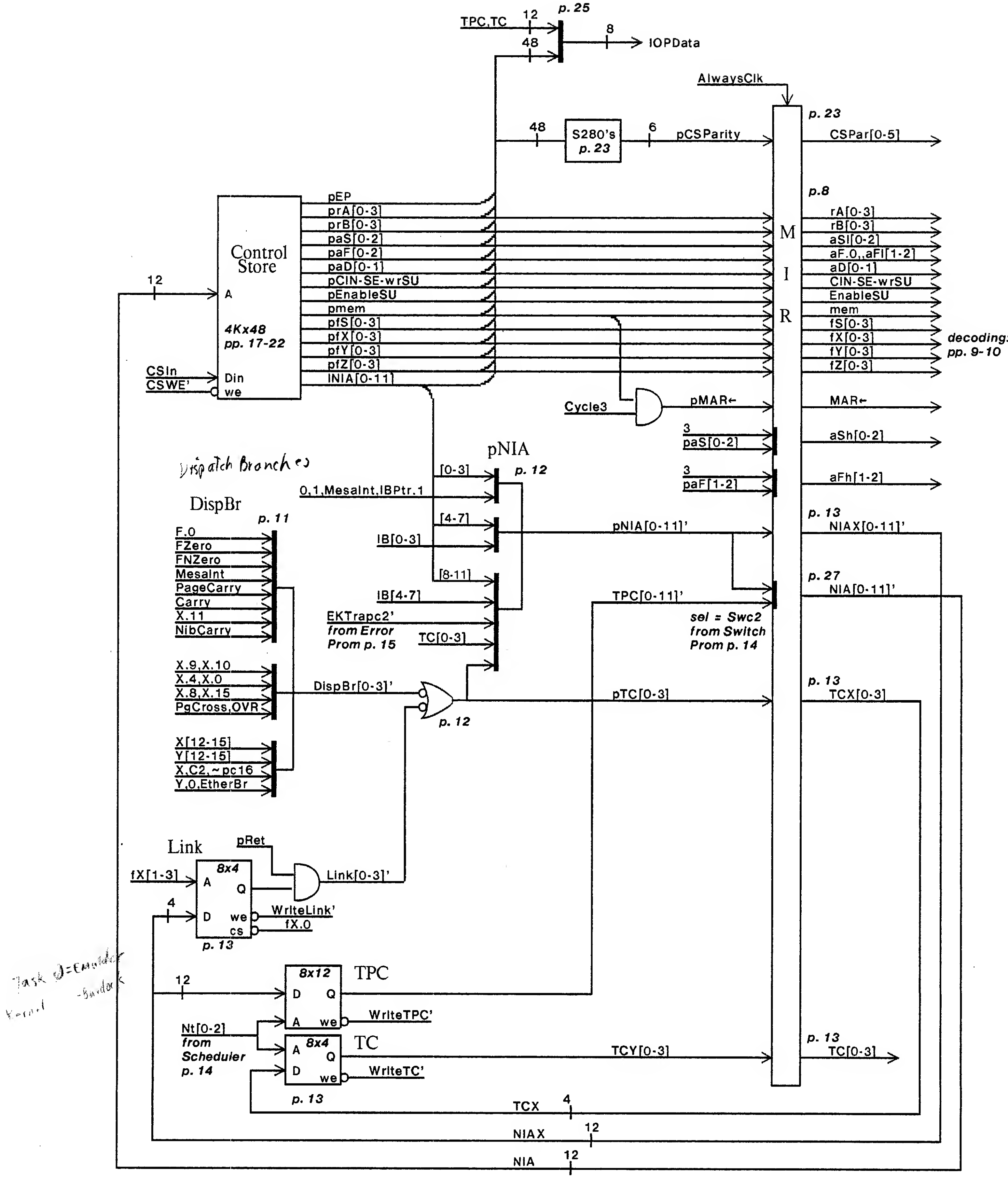


NOTE: C1-C65. CAP., CERAM, 50V , .10UF , PART NO. 702W05218

XEROX ED	Project Dandelion	Filter Capacitors	File pLionHead29.sil	Designer Lin	Rev M	Date 8/23/80	Page 29

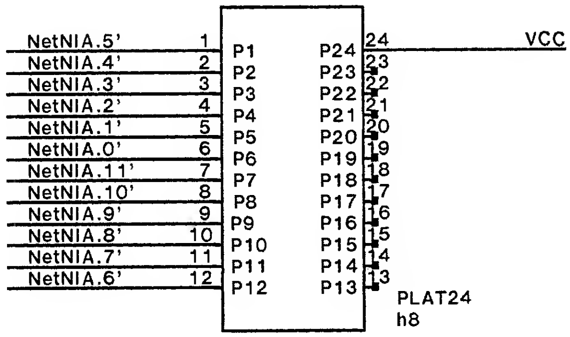


XEROX SDD	Project Dandelion	Block Diagram I	File LionHead38.sily	Designer Garner	Rev K	Date 30 Oct 80	Page 38

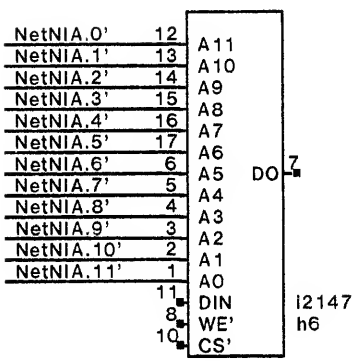


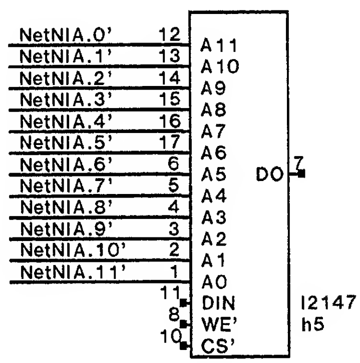
XEROX SDD	Project Dandelion	Block Diagram II	File LionHead39.sily	Designer Garner/ W.H.	Rev K	Date 30 Oct 80	Page 39

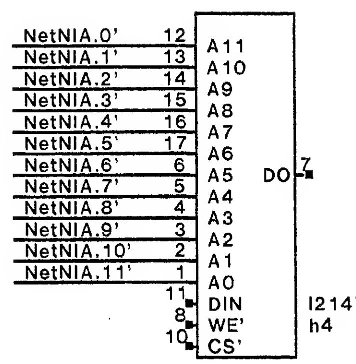
This diagram defines the NetNIA.wl wirelist. This list should be wired before LionHead.wl.
Except for the first entry in each net, each node should be welded off center and towards the closest edge of the board

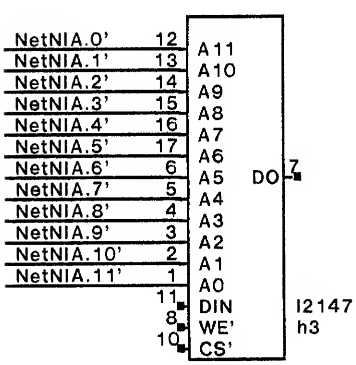


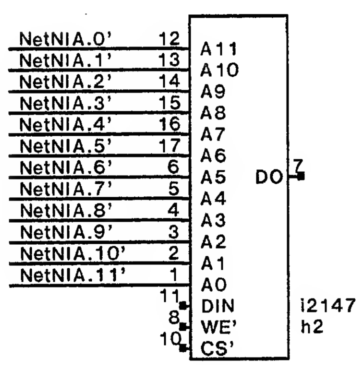
Just cut the ground connection (which is really a NetNIA line),
the LionHead wire list will cut the VCC connection.
(The LionHead wire list should not try to cut the GND again,
since it will have been connected to NetNIA.11')

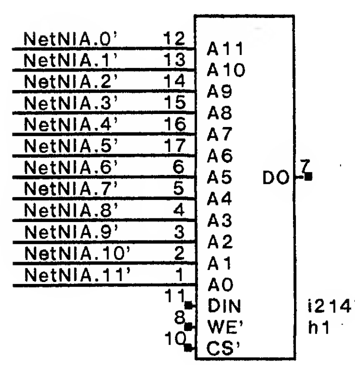












Rev A to Rev B (9 Oct 79)

1. Added timing info to all pages. Divided page 14 into 14 and 15, renumbering original 15-25.

Page 2: a. 1K pullup pack changed to 22-330 resistor pullup/pulldown. Ground to P8.
b. S09 changed to S03. Bits into Q ends inverted now. CIN-SE-wrSU' and pc16' necessary for CIN-SE.

Page 4: a. stackP read (instead of NstackP) onto X-bus (allows stackP in arithmetic operations).
b. RH[0-3] moved to d17.

Page 5: a. IBProm changed: SelIB0' and SelIB1' are now used to immediately select either IB[0] or IB[1].
IB←' input removed and replaced with EKErrc2 (to cancel GoodIBDispc2, instead of at the pNIA S64's.
Interchanged IBPtr.0 and IBPtr.1, deleted IBPtr.1'.

Page 6: a. Changed pin4 of f19 from Y.4 to Y.3
b. Interchanged fZ.3 and fZ.2 on 25S10's

Page 7: a. Changed ErrInt Status to ErrIBPtr, i.e. subsituted IBPtr.0 for MesaInt' and IBPtr.1 for CSParErr'.

Page 8: a. Changed mem from pin 14 to pin 113.
b. Moved CIN-SE-wrSU from c9 to b9, creating CIN-SE-wrSU'. paF.0 took CIN's place. aD.0 was moved to aD.1, and aD.1 to aF.0
c. Changed MAR← to MAR←', discontected MAR← from backplane.

Page 9: a. On b11(fZNorm), changed AlwaysNI' to IBPtr←0'; fS.2 to fS.2'; and moved all outputs up one position.

Page 10: a. Added S04 inverter for aD.0', moved RH← to page 16.
b. Changed S20's to S08 + S10's, opening up an S10 for use.

Page 11: a. Replaced Cycle1 test with CSParErr and NibCarry test with MesaInt.

Page 12: a. at pNIA[0-3] changed pin 6 from GND to HIGH (to distinquish no interrupt, empty buffer from error trap at 0)
b. Rearranged pNIA[8-11] S64 inputs: EKErrc2 should have zeroed the dispatch/branch bits also.

Page 13: a. Moved Link.3' connection to pullup pack since it is now 220-330 Pullup-down.
b. Changed NIA's SB inputs from Swc3 to Swc2.

Page 14: a. Enlarged Schedule Prom, adding RefReq' input. Pullup connections to requests from Options board.
b. Changed all inputs to SwitchProm (see programs).

Page 15: a. MemCSErrProm renamed CSIntProm since MemErr moved out to S08 and MesaInt moved in.
b. StackVirtErrProm renamed StackVirtProm, ClrIntErr' Input not needed.
c. ErrorProm inputs changed: Nt = Emu to Ct = Emu.
d. KEProm renamed KernPC16Prom since MesaInt moved out. KernReq' an input now.

Page 16: a. WriteIB qualifier changed from S08 to S260 with ppCLK--reduced IB's large hold time.
b. WrTPCLow Inverted, RH← moved here.
c. Detection of Low bank changed to S260 (freeing up and S02 and S08).

Page 25: a. LS251 inputs rearranged so read data is identical to write data format.

Rev B to Rev C (17 Dec 79)

Page 2: a. Changed S1 of R Shift Ends from Cycle' to Shift' (to accomodate new fY = Cycle). Switched inputs to mux.
c. Added S86 PageCross.

Page 3: a. Added LS257 AltUAddr allowing low SU address to come from Y bus.

Page 5: a. EKErrc2 input to IBProm changed back to IB←'. PgCarryDly replaced by AllowMDR←. IBFront inverted.

Page 8: a. rB's S74's replaced by S374.
b. aS, aFh's S374 + S32 replaced by 25S09
c. Added MarPgCross' & AllowMDR← (to pin 11 backplane)

Page 9: a. Added Cycle to fY; AltUAddr to fY; PushNT to fZ
b. Inverted S.2 and S.3 values used to select IOIn.

Page 10: a. With new fY cycle, changed sh's S00 to S10; changed Pop's S00 to S86; With PushNT, changed Push's S00 to S10. changed MapRef's S00 to S86.

Page 11: a. Changed 4:1 mux for DispBr[0-1]' to 2:1 S258.
b. XC2npcDisp ← IODispA[2-3]; YIODisp ← IODispB[2-3]; IODisp ← XpcDisp; [] ← XwdDisp; PgCrossBr ← PgCarryBr; NibCarryBr ← DirtyBr;

Page 12: a. Interchanged TC's and IB's contection to pNIA[8-11], so that IB is blocked by EKTrapc2 (renamed from EKErrc2).
b. EKTrapc2 doesn't zero pNIA.11 (reduces loading on EKTrapc2 below max)
c. pTC.2 changed from S00 to S10 adding MarPgCross'

Page 14: a. SpareReq' added to ScheduleProm & Pullups. Task register enabled by Cycle2' now.
b. Nt = Emu from Pt = Emu established.
c. Waitc2' changed to Wait in SwitchProm.

Page 15: a. MemErr not gated by Pt = Emu any longer. Connected to ErrorProm instead.
b. CSParErr connected to BP pin 37.

Page 16: a. WriteTC = pAlwaysClk AND Cycle1 AND TCWait'. TCWait' added beyond Waitc3.
b. IOPWait gated with cycle1 now (so Stop correctly works).

Page 24: a. SwTAddr, SwTAddr', & IOPWait temporarily synched by Clock until IOP card does it.

Rev C to Rev D (1 Feb 80)- Rev D submitted for 1st etch.

Page 1: Added PDT-Y bus for testability; Moved the 2901's to free up 3 board positions.

Page 2: Q.0 & Q.15 pullups now 1K; Cin's pullup is 100 ohms; S03 replaced with S38 (used on HSIO board)

Page 5: IBFront changed to LS374; added PDT for testability; h17 now LS374.

Page 7: Byte inverted => D & B inputs interchanged on S257.

Page 8: rA & rB swapped (for layout); S08 to LS08; HIGH-a to PU

Page 9: fYNorm, fZNorm updated; fX moved to c13.

Page 10: PushY' added to Push; Refresh is now RefreshY or RefreshZ; S32 to LS32; XBus←IB' now S20 (so MAR←IB works)
Byte S08 changed to S00; fZHigh eliminated

Page 11: S258 changed to LS158; PgCrOvDisp, YOddBr added; PageCross changed back to PageCarry.

Page 12: EKTrap zeros pNIA.11 (again).

Page 13: Am29700 changed to Am29701 and LS32 gates output.

Page 14: Tasks reg changed to LS374; PDT added for testability.

Page 15: Now Switchweld & PC version of page 15: Sw version does not have CSPar.4 & CSPar.5 to CSIntProm--now a F93453
PDt added for testability; h17 & g15 now LS374; PopX' replaces Pop & Popz' added to StackProm.

Page 16: WrIBFront and C2Clk qualifiers added; WrTPCLow no longer inverted.

Page 23: Now Switchweld & PC version of page 23: Sw version uses 93S48 12-input parity chips & PC version uses S280's.

Page 24: IOPWait, SwTAddr, SwTAddr' no longer temporarily clocked; IOPBus renamed IOPData; LS241 replaced with LS244.

Page 25: IOPBus renamed IOPData

Page 26: LS244 added for testability

Page 27: Discrete page for PC & Sw versions--PC versions includes fuse & supply caps.

Page 28: Unused parts

Rev D to E (13 Feb 80) -- version actually used for 1st PC etch

5 SelectIB0 swapped with SelectIB1' (name change only)
9 ←IOInSp1' changed to ←KTest' (name change only)
10 changed S86 to S00 which generates Pop since both PopX' and PopZ' can be active now..
17-22 For the PC version only, the NIA.# a', NIA.# b', NIA.# c', NIA.# d', NIA.# e', and NIA.# f' nets were replaced by NetNIA.#'
This change was made for PC only.
p27 NIA.#' was swapped with NetNIA.#' on the resistors so the error messages would go away...
28 S86 now spare instead of S00.

Rev E to F (15 March 80) -- updates made to 1st PC etch.

2 S04 at c14d used to form FNZero
5 S260 at d16b now used to generate IBEmptyErr. IBProm now rev E.
8 LS08 gate producing BrMAR← from MAR← & IgnorePgCr added.
9 MapRefZ' deleted and MapRefY' added. changed name of DOAData←' & DOBData←' to DCtlFifo←' & DBorder←'
IgnorePgCr' replaces MapRefZ'
10 MapRefZ' replaced with MapRefY'
11 IODisp replaced with XwdDisp. YOddBr replaced with NZeroBr.
12 IBPtr.1 replaced with IBPtr.0 (due to renumbering ibPtr states).
13 Per Testability Review, added PDT-tpc. Link pRet enable kludge made worse by ading LS32 producing pRet'.
15 IBEmptyErr replaces Pt = Emu in ErrorProm. MemErrC3 now gated with a LS08. ErrorProm now Rev D.
Moved 374 clocks and enables to page28.
16 WriteIB now comes from S02 at a19d and LS32 at e14b (an ADDITIONAL PART).
17-22 Per Testability Review, PDT-CS-a and PDT-CS-b were added to the control store chips.
26 Per Testability Review, PDT-tpc, PDT-CS-a and PDT-CS-b added to LS244.
28 Moved Junk 374 clocks & enabled here, including the one for f9 (WHICH WAS FORGOTTEN in Rev E).
Only one LS08, one S04 spare now. two LS32's gates spare.

Rev F to G (15 April 80) -- updates made to 1st etch.

LS08 at b13 changed to S08; LS374 at g15 changed to S374
8 pAllowDMR← added: No mem write on IBEmtpyErr OR MarPgCross; IgnorePgCr eliminated
9 IgnorePgCr' eliminated.
12 pNIA.3' fed by IBPtr.1 instead of IBPtr.0 (above fix wrong)
15 StackVirtProm & ErrorProm now Rev E.
p27,s27 NetNIA', AlwaysClk, & GND connected to 14 pin plat at i5 for Logic Analyzer
28 S04 at c14f no longer unused.
49y Printer registers
51y S08, S374 parts change

Rev G to H (14 May 80) -- updates made to 1st etch

163 test points were added
13 PDT-tpc connected to g9.2 (CS'). NIA moved to page 27.

Rev H to I (July 80) -- changes made by Richard Johnson (ED) to 1st etch layout (hand changes) plus other name/part changes
-- There were 8 Rev I boards manufactured. They require fixes to the incorrect changes below described.

Am29701's replaced by Am27S07's
4 half of S241 at c18 and half S241 at d18 changed (inadvertently) into an S244. e17e-h swapped with a-d.
5 g17(v16)b-e reverse ordered. g16(v44)f-i reverse ordered. f16(v43)f-i reverse ordered, LS374 changed to S374.
6 half of S241 at c18 and half of S241 at d18 changed (inadvertently) into an S241-like chip with 2 EN inputs.
7 f18c got swapped inputs of f18d, f18d got swapped inputs of f18e, f18e got swapped inputs of f18c.
Swapping inputs to the S257 multiplexer was (moderately) incorrect!!
9 XOData,XCtl,XIData,XStatus,IOOutSp1,IOOutSp2,IOOutSp3,PStatus,IOInSp2 renamed
13 Swapped inputs of i4.
14 SpareReq' renamed EORound
24 e8e-h reverse ordered. i13e interchanged with i13g.
27 i5.14 replaced with AlwaysClk-c

Rev I to J (24 Aug 80) -- changes made for etch 1.5 (Never built)

4,6 u146 & u118 (c18,d18) reconnected as in Rev. H
7 u142 (f18) reconnected as in Rev. H
10 u70b connected in order to invert Refresh (This change was not made on the MCtl card as directed)
27 R21 connected to u138.1 instead of SpareReq' (renamed to EORound)

Rev J to K (30 Oct 80) -- etch 2 definition

5 IBProm (HM7649) split into two F93453's.
14 XReq' renamed EReq'
24 LS244 @u1 (CSDataIn) changed to LS240 (to prevent ringing). -- requires change in IOP Kernel

Rev K to L (3 Dec 80) -- changes made to 2nd etch

3, 5, 9, 13, 15, 23 Added test points according to D. Adams, ES

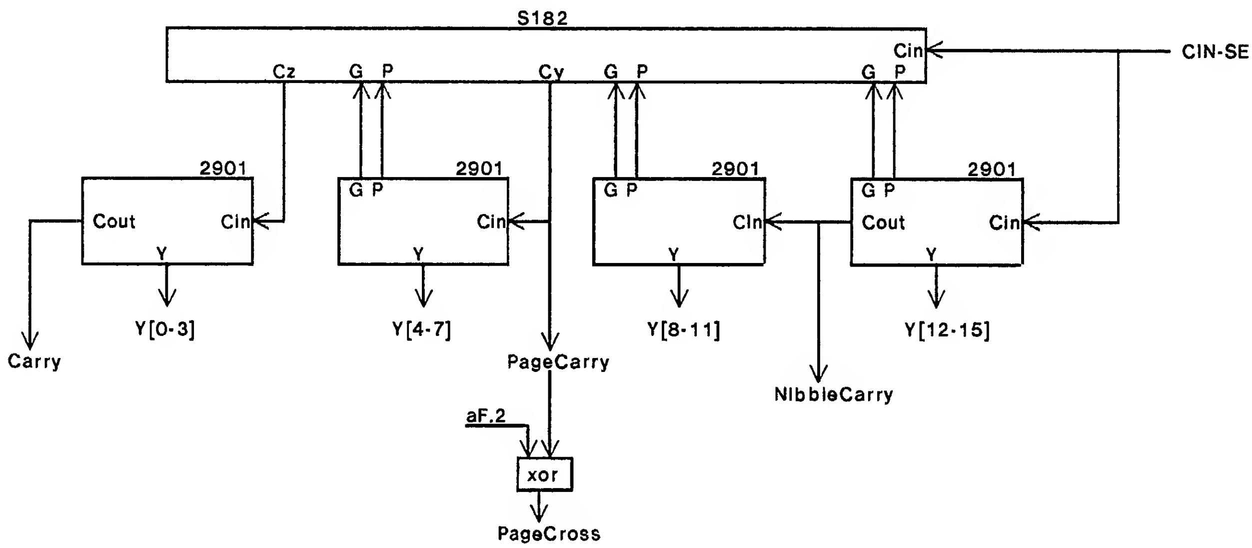
Rev L to M (2 April 81) -- etch 3 definition

10 u70b removed so that Refresh is no longer inverted (reality wins over politics)

YH.,Y Bus MAR← timing: For high-half ALU, operation is 0 or B.

17[3] Clk -> rB 35 rB -> YH 5 YH bus 57[3] = 60 nS	80 nS (See Ybus logic timing)	59 disable X 67 D -> Y 126 nS	59 disable X (See X bus timing) 42 D -> Y (See Y bus arithmetictiming) 101 nS

Y bus Arith timing: Ripple carry is used in low half of ALU, and lookahead in high half.



X bus arithmetic: Ybus ← XBus + A

x	Xbus time
30	D -> G,P[4-11]
7[2]	G,P -> Cin.7, Cin.3
25	Cin -> Y[0-7]
10	Y bus
72[2]	= (74 + x) nS

x	Xbus time
32	D -> Cout[12-15]
25	Cin -> Y[8-11]
10	Y bus
(67 + x) nS	

x	Xbus time
32	D -> Y[12-15]
10	Y bus
(42 + x) nS	

Register Arithmetic: Ybus ← A + B

max(109, 95)

17[3]	↑ -> rB
45	rB -> G,P
7[2]	G,P -> Cin.7, Cin.3
25	Cin -> Y[0-7]
10	Y bus
104[5]	= 109 nS

48	↑ -> CIN-SE (see p. 2)
11[1]	S182 Cin -> Cin.7, Cin.3
25	Cin -> Y[0-7]
10	Y bus
94[1]	= 95 nS

max(105, 99)

17[3]	↑ -> rB
50	rB -> Cout[12-15]
25	Cin -> Y[8-11]
10	Y bus
102[3]	= 105 nS

48	↑ -> CIN-SE (see p. 2)
16	Cin[12-15] -> Cout
25	Cin -> Y[8-11]
10	Y bus
99	nS

max(83, 80)

17[3]	↑ -> rB
50	rB -> Y[12-15]
10	Y bus
77[3]	= 80 nS

48	↑ -> CIN-SE (see p. 2)
25	Cin -> Y[12-15]
10	Y bus
83	nS

Y bus Logic Timing:

Xbus Logic
Ybus ← Xbus .or. 0

x	X bus
32	D -> Y
10	Ybus
(42 + x) nS	

Register Logic:
Ybus ← A .or. B

17[3]	↑ -> rA, rB
50	rB -> Y
10	Ybus
77[3] = 80 nS	

A pass around (aD=2):
Ybus ← A

17[3]	↑ -> rA
40	rA -> Y (via A bypass)
10	Ybus
67[3] = 69 nS	

X bus Source timing:

Xbus ← A LRotn {A bypass}

69	↑ -> Y (see above)
12	25S18 prop
10	Xbus
91 nS	

(Xbus to Ybus time = 22 nS)

Disable = 56 nS

		Disable
Xbus ← (A or B) LRotn	80 + 22 = 102 nS	Disable = 56 nS
Xbus ← (A + B) LRotn	109 + 22 = 131 nS | 105 + 22 = 127 nS | 83 + 22 = 105 nS	
Xbus ← SU	75 nS (see p. 3)	Disable = 58 nS
Xbus ← RH[B]	74 nS (see p. 4)	Disable = 59 nS
Xbus ← IB	59 nS (see p. 5)	Disable = 49 nS
Xbus ← MD	97 nS max	Disable = 17[3] + 3 (BP) + 5 + 15 + 10(Xubs) = 53 nS
Xbus ← 4-bit constant	50 nS (see p. 7)	Disable = 50 nS
Xbus ← 8-bit constant	56 nS (see p. 7)	Disable = 49 nS
Xbus ← ErrIBStkP	59 nS (see p. 4)	Disable = 59 nS
Xbus[0-7] ← 0	55 nS (see p. 7)	Disable = 55 nS

Xbus ← IOIn

34	↑ -> ← IOIn'	34	↑ -> ← IOIn'
3	backplane	3	backplane
15	S240 EN' to X	18	S374 EN' to X
10	Xbus	10	Xbus
62 nS		65 nS	

Disable = 59 nS

External Register Write Setups:

SU Write Setup (SU ← Ybus):

5[1]	F93422 data setup (from beginning of write pulse)
39	WE pulse width
44[1] = 45 nS	

RH Write Setup (RH ← Xbus): 36 nS (see p. 4)

IB Write Setup (IB ← Xbus): 37 nS (see p. 5)

IOOut Write Setup (IOOut ← Xbus): equals setup time of receiving reg.
data setup for LS374/LS273 = 20[2] = 22 nS

Dispatch/Branch Condition Bits Setup:

7[1] S151/S258 in -> DispBr
5 DispBr -> pTC
6[1] pTC -> pNIA
5[1] 25S09/S374 setup
23[3] = 26 nS

D-input Setup Times:

Logic
B ← Xbus or 0
B ← Xbus or A

40 nS

Logic & Branch
B ← Xbus .xor. A, ZeroBr

32 D -> F.0, F = 0
26 DispBr setup
58 nS

Logic & YDisp
B ← Xbus .xor. A, YDisp

32 D -> Y
10 Ybus
26 DispBr setup
68 nS

Logic & Single-Bit Shifting
B ← Xbus .or. A, LShift1

35 D -> R.3
15 RAM3 setup
50 nS

Logic & Single-Bit Rotates
B ← Xbus .or. A, LRot1

35 D -> R.15
9[1] S253 in to R.0
15 RAM0 setup
59[1] = 60 nS

Logic & Double-Bit Shift/Rotate
B ← DRShift1 B

35 D -> R.15
20 S38 in to Q.0
10 RAM3 setup
65 nS

	Xbus[0-7]	Xbus[8-11]	Xbus[12-15]
Register Arithmetic B ← Xbus + A	30 D -> G,P 7[2] G,P -> Cin.3, Cin.7 35 Cin setup 72[4] = 74 nS	32 D -> Cout[12-15] 35 Cin.11 setup 67 nS	40 nS (Logic setup)
Register Arithmetic & ZeroBr B ← Xbus + A, ZeroBr B ← Xbus + A, NZeroBr	30 D -> G,P 7[2] G,P -> Cin.3, Cin.7 30 Cin -> F = 0 26 DispBr setup 93[2] = 95 nS max(95, 86, 58)	32 D -> Cout[12-15] 30 Cin -> F = 0 26 DispBr setup 88 nS	58 nS (Logic&Branch) Add 5 nS for NZeroBr
Register Arithmetic & NegBr B ← Xbus + A, NegBr	22 Cin -> F.0 = 95-8 = 87 nS		
Register Arithmetic & OvBr B ← Xbus + A, OvBr	25 Cin -> Ovr = 95-5 = 90 nS		
Register Arith & Carry branches B ← Xbus + A, NibCarryBr B ← Xbus + A, PgCrossBr B ← Xbus + A, CarryBr	30 D -> G,P 7[2] G,P -> Cin 16 Cin -> Cout.0 26 DispBr setup 79[2] = 81 nS (CarryBr)	30 D -> G,P 7[2] G,P -> PgCarry 11[1] PgCarry -> PgCross 26 DispBr setup 74[3] = 77 nS (PgCrossBr) (MarPgCrossBr = 75 nS)	max(58 + x, 90) nS 32 D -> Cout 26 DispBr setup 58 nS (NibCarryBr) 48 ↑ -> CIN-SE 16 Cin -> Cout 26 DispBr setup 90 nS
Arithmetic & YDisp B ← Xbus + A, YDisp	Timing for X[0-7] does not affect YDisp		68 nS (Logic&YDisp)
Arithmetic & Single-Bit Shifting B ← Xbus + A, RShift1	30 D -> G,P 7[2] G,P -> Cin.3, Cin.7 35 Cin -> R.3 15 RAM3 setup 87[2] = 89 nS	30 D -> Cout[12-15] 35 Cin -> R.3 15 RAM3 setup 80 nS	50 nS (Logic&Shifting)
Arithmetic & Singl-Bit Rotating B ← Xbus + A, RRot1	89 + 10 = 99 nS	80 + 10 = 90 nS	60 nS (Logic&Rotating)

Arithmetic & Double-Bit Shift/Rotate
B ← DARShift1 B

30 D -> G,P
7[2] G,P -> Cin
16 Cin -> Cout.0
9[1] S253 to R.0
15 RAM0 setup
77[3] = 80 nS

R Register Cycle Times

Operation	bits[0-7]	bits[8-11]	bits[12-15]
Register Logic B ← A .and. B	17[3] ↑ -> rA 60 rA setup 77[3] = 80 nS		
Register Logic & Branch B ← A .xor. B, ZeroBr, B ← A .xor. B, NegBr,	17[3] ↑ -> rA 55 rA -> F = 0 26 DispBr setup 98[3] = 101 nS	17[3] ↑ -> rA 50 rA -> F.0 26 DispBr setup 93[3] = 96 nS (NegBr)	
Register Logic & YDisp B ← A .xor. B, YDisp,	80 Ybus ← A .xor. B 26 DispBr setup 106 nS		
A Bypass & YDisp [] ← A, YDisp	69 Ybus ← A 26 DispBr setup 95 nS		
Register Logic & Shifting B ← A .or. B, LShift1	17[3] ↑ -> rA 55 rA -> R.3 15 RAM3 setup 87[3] = 90 nS	17[3] ↑ -> rA 55 rA -> R.3 20 S38 to Q.0 10 Q0 setup 102[3] = 105 nS (DRShift1)	30 ↑ -> Q.0 9[1] S253 to R.15 15 RAM3 setup 54[1] = 55 nS (DLShift1)
Register Logic & Rotating B ← A .or. B, LRot1	17[3] ↑ -> rA 55 rA -> R.3 9[1] S253 in to R.0 15 RAM0 setup 96[4] = 100 nS		

Operation	bits[0-7]	bits[8-11]	bits[12-15]
Register Arithmetic B ← A + B	max(109, 98) 17[3] ↑ -> rA 45 rA -> G,P 7[2] G,P -> Cin.3, Cin.7 35 Cin setup 104[5] = 109 nS 48 ↑ -> CIN-SE 11[2] CIN-SE -> Cin.3(S182) 35 Cin setup 94[2] = 98 nS	max(105, 99) 17[3] ↑ -> rA 50 rA -> Cout[12-15] 35 Cin.11 setup 102[3] = 105 nS 48 ↑ -> CIN-SE 16 CIN-SE -> Cout[12-15] 35 Cin.11 setup 99 nS	80 nS (Reg Logic)
Register Arithmetic & Branch B ← A + B, ZeroBr B ← A + B, NZeroBr B ← A + B, NegBr B ← A + B, OvBr B ← A + B, CarryBr, B ← A + B, PgCrossBr, B ← A + B, NibCarryBr,	17[3] ↑ -> rA 45 rA -> G,P 7[2] G,P -> Cin.3, Cin.7 30 Cin -> F = 0 26 DispBr setup 127[5] = 132 nS (ZeroBr) 22 Cin -> F.0 = => 124 nS (NegBr) 25 Cin -> OVR = => 127 nS (OvBr) 16 Cin -> Carry = => 118 nS (CarryBr)	17[3] ↑ -> rA 30 rA -> Cout[12-15] 30 Cin -> F = 0 26 DispBr setup 103[3] = 106 nS (ZeroBr) 22 Cin -> PgCarry 11[1] PgCarry -> PgCross 33[1] = 34 nS = => 110 nS (PgCrossBr)	101 nS (Logic&Branch) Add 5 nS for NZeroBr 48 ↑ -> CIN-SE 16 CIN-SE -> NibCarry 26 DispBr setup 90 nS (NibCarryBr)
Arithmetic & YDisp B ← A + B, YDisp	Timing for X[0-11] does not affect YDisp		104 nS (Logic&YDisp)
Arithmetic & Shifting B ← A + B, RShift1	17[3] ↑ -> rA 30 rA -> G,P 7[2] G,P -> Cin.3, Cin.7 35 Cin -> R.3 15 RAM3 setup 107[5] = 112 nS	17[3] ↑ -> rA 30 rA -> Cout[12-15] 35 Cin -> R.3 15 RAM3 setup 97[3] = 100 nS	90 nS (Logic&Shifting)
Arithmetic & Rotating B ← A + B, RRot1	112 + 10 = 122 nS	100 + 10 = 110 nS	100 nS (Logic&Rotating)

X Operation	D setup	X Source: SU	MD	* RH	* Nibble	* Byte	* IB	* ErrIBStkP	IOIn	A LRotn	(A .or. B) LRotn	(A+B) LRotn
X←		75	97	74	50	56	59	59	63	91	102	
max (59, X←)		75	97	74	59	59	59	59	63	91	102	131 127 105
B←X or A	40	115	137	114	99	99	99	99	103	131	—	—
[]←X or A, ZeroBr	58	133	155	132	117	117	117	117	121	149	—	—
[]←X or A, NZeroBr	63	138	160	137	122	122	122	122	126	154	—	—
[]←X or A, NegBr	58	133	155	132	117	117	117	117	121	149	—	—
[]←X .or.A, YDisp	68	143	165	142	127	127	127	127	131	159	—	—
B←X .or. A, LShift1	50	125	147	124	109	109	109	109	113	—	—	—
B←X .or. A, LRot1	60	135	157	134	119	119	119	119	123	—	—	—
MAR←X .or. A	78	153	—	152	137	137	137	137	141	—	—	—
MDR←X .or. A	45	120	—	119	104	104	104	104	108	—	—	—
SU←X .or. A	87	—	184	161	146	146	146	146	150	—	—	—
IOYOut←X .or. A	64	139	161	138	123	123	123	123	127	—	—	—
B←X + A	74 66 40	149 141 115	171 163 137	143 140 114	133 125 99	133 125 99	133 125 99	133 125 99	137 129 103	165 157 131	—	—
[]←X + A, ZeroBr	95	170	192	169	154	154	154	154	158	186	—	—
[]←X + A, NegBr	87	162	184	161	146	146	146	146	150	178	—	—
[]←X + A,OvBr	90	165	187	164	149	149	149	149	153	181	—	—
[]←X + A,NibCarry	58	133	155	132	117	117	117	117	121	149	—	—
[]←X + A,PgCarryBr	65	140	162	139	124	124	124	124	128	—	—	—
[]←X + A,PgCrossBr	77	152	174	151	136	136	136	136	140	168	—	—
[]←X + A,CarryBr	81	156	172	155	140	140	140	140	144	171	—	—
[]←X + A,YDisp	68	143	165	142	127	127	127	127	131	159	—	—
B←X + A, RShift1	89 80 50	164 155 124	186 177 147	163 154 124	148 139 109	148 139 109	148 139 109	148 139 109	152 143 113	—	—	—
B←X + A, RRot1	99 90 60	174 165 135	196 187 157	173 164 134	158 149 119	158 149 119	158 149 119	158 149 119	162 153 123	—	—	—
MAR←X + A	- 78 78	153	—	152	137	137	137	137	141	—	—	—
MDR←X + A	77 70 45	152 145 120	—	151 144 119	136 129 104	136 129 104	136 129 104	136 129 104	135 133 108	—	—	—
SU←X + A	119 112 87	—	216 209 184	193 186 161	178 171 146	178 171 146	178 171 146	178 171 146	182 174 150	—	—	—
IOYOut←X + A	80 73 48	155 148 123	177 170 145	154 147 122	139 132 107	139 132 107	139 132 107	139 132 107	143 136 111	—	—	—
[] ← X, XDisp	32	107	129	106	91	91	91	91	94	123	134	163 159 137
RH ← X	36	111	133	110	95	95	95	95	99	127	138	167 163 146
IB ← X	37	112	134	111	96	96	96	96	100	128	139	168 164 148
IOXOut←X (LS374)	22	97	117	96	79	79	79	79	83	111	122	153 149 127

* Timing for bits[0-7] of these sources is that of Nibble
The 3 numbers for arithmetic operations correspond to bits[0-7], bits[8-11], & bits[12-15], respectively.
stackP← has timing of the slow IOYOut.

Y Operation	setup	Y Source: A .or. B	A (bypass)	A + B
Y ←		80	69	109 105 83
MAR ← *	36 11 36	116 91 116	105 80 105	114 116 119
MDR ←	3	83	72	112 108 86
SU ←	45	125	114	***154*** ***150*** 128
stackP ←	6	86	75	115 112 89
[]← , YDisp	32	112	101	121
Uaddr[4-7]←	15	95	84	124 120 98
IOYOut← (S374)	6	86	75	115 112 89

* Bits[0-7] have timing of Y ← (B .or. 0), except in the A bypass case.

The 3 numbers for arithmetic operations correspond to bits[0-7], bits[8-11], & bits[12-15], respectively.

X bus Loading & Estimated Capacitance

(for X[12-15] since these bits have the greatest loading & length)

Capacitances are based on experimental measurements (see p. 55)

Source	Sink	Part	Source Drive	Sink Load	Capacitance (pF)
	D-input	IDM2901A-1		.4/.18	4
	RH	Am27S07		.2/.125	4
	IB	S373		1/.125	4
	XDisp	S151		1/1	4
	XLDisp	S151		1/1	4
	"HSIO"	S241		1/.2	8
	"Option"	S241		1/.2	8
	IOPOData	LS374		.4/.2	4
	IOPCtl	LS273		.4/.2	4
SU		93422	104/4	1/.025	5
LRotn		Am25S10	130/10	1/.025	9
ErrIBStkp		S240	60/32	1/.025	11
RH		S241	60/32	1/.025	11
IB		S257	130/10	1/.025	5
Nibble		S241	60/32	1/.025	11
MD		S240	60/32	1/.025	11
IOPIData		S374	130/10	1/.025	5
IOPStatus		S240	60/32	1/.025	11
XIData		S374	130/10	1/.025	5
XStatus		S240	60/32	1/.025	11
KIData		S374	130/10	1/.025	5
KStatus		S240	60/32	1/.025	11
KTest		S240	60/32	1/.025	11
MStatus		S240	60/32	1/.025	11
Min Source Drive		S240/93422	60/4		
Total Sink Load				22/3.7	

Total Component Capacitance	177 pF
Etch @ 50" (CP=20, HSIO=10, MCtl=8, Opt=5, IOP=4, BP=3)	150 pF
Total X bus Capacitance	327 pF

Table Entries: High U.L./ Low U.L.
1 High U.L. = 50 uA
1 Low U.L. = 2.0 mA

Y[0-7] Bus loading

Source	Sink	Part	Source Drive	Sink Load
Y-output		IDM2901A-1	32/10	
	LRotn	Am25S10		1.5/1.5
	SU	93442		.8/.15
Y.4	MAR	S253		3(1/1)
	MDR	S373		1/.125
	MCtl	S138		1/1
	DCtlFifo	S373		1/.125
	DBorder	LS374		.4/.2
Total Sink Load				8.7/6.1

Y[8-15] Bus loading

Source	Sink	Part	Source Drive	Sink Load
Y-output		IDM2901A-1	32/10	
	LRotn	Am25S10		1.5/1.5
	SU	93442		.8/.15
	stackP	25S09		1/1
	AltUAddr	S257		1/1
Y.12	MAR	S253		2(1/1)
	MDR	S373		1/.125
	MCtl	LS374		.4/.2
	DCtlFifo	S373		1/.125
	DBorder	LS374		.4/.2
	YDisp	S151		1/1
Total Sink Load				10.5/7.4

Table Entries: High U.L./ Low U.L.

1 High U.L. = 50 uA
1 Low U.L. = 2.0 mA

PC version:	#	I_{typ}	I_{total}	
IDM2901A-1	4	160	640	
Am27S07	7	75	525	
AM25S09	7	75	525	
AM25S10	4	60	240	
i2147L	48	100	4800	
F93422	4	95	380	
F93427	2	85	170	
F93453	4	120	480	
HM7649	1	120	120	
SN74S00	4	15	60	
SN74S02	1	22	22	
SN74S04	2	23	46	
SN74S08	1	25	25	
SN74S10	2	12	24	
SN74S20	1	6	6	
SN74S38	1	32	32	
SN74S51	1	11	11	
SN74S64	4	8	32	
SN74S86	1	50	50	
SN74S138	8	49	392	
SN74S151	3	45	135	
SN74S175	1	60	60	
SN74S182	1	69	69	
SN74S240	1	90	90	
SN74S241	3	108	324	
SN74S253	1	55	55	
SN74S257	4	52	208	
SN74S260	1	22	22	
SN74S280	6	67	402	
SN74S373	2	105	210	
SN74S374	7	90	630	
SN74LS32	3	4	12	
SN74LS158	3	5	15	
SN74LS244	4	20	80	
SN74LS251	8	7	56	
SN74LS283	1	20	20	
SN74LS374	1	27	27	
	156		11.0 Amps	(70 mA/chip)

Stichweld only:				
Am93S48	4	57	228	
	154		10.8 Amps	(70 mA/chip)

1 10 20 30 40 50 | 51 60 70 80 90 100
101 150 | 151 200

Am29701 can be used instead of Am27S07

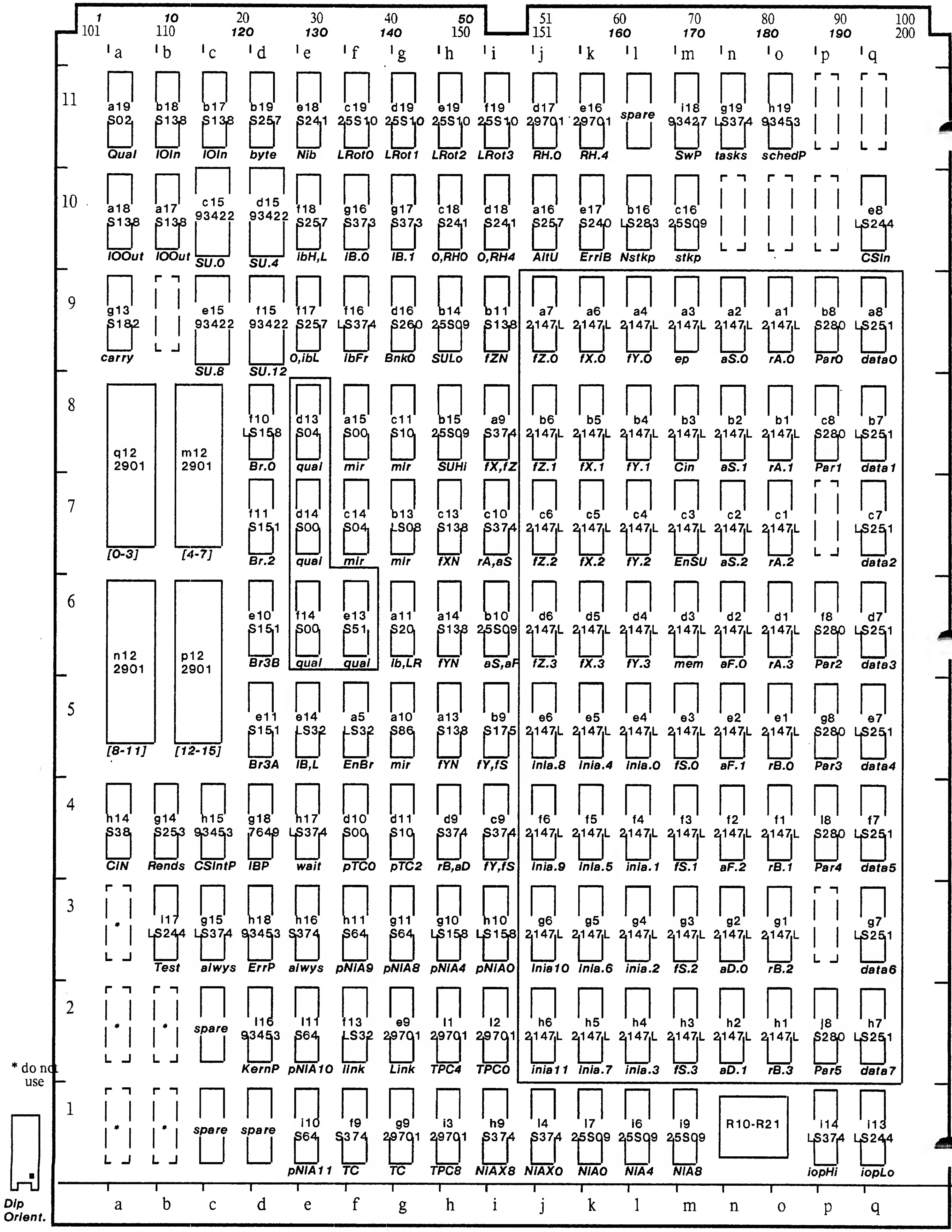
	a	b	c	d	e	f	g	h	i	
19	S02 pWait,pAlws Pt, WrIB	S257 0, Byte	25S10 LRotn.0	25S10 LRotn.1	25S10 LRotn.2	25S10 LRotn.3	LS374 Tasks	F93453 SchedProm	LS244 CS-IOP Recv	19
18	S138 IOOut	S138 IOIn	S241 0,RH[0-3]	S241 0,RH[4-7]	S241 Nibble, Pt	S257 IbLow, ibHigh	HM7649 IBProm	F93453 ErrorProm	F93427 SwProm	18
17	S138 IOOut	S138 IOIn	F93453 StackVirtErr	AM27S07 RH[0-3]	S240 ErrIBStkp	S257 0, IbHigh	S373 IB[1]	LS374 WaitClks	LS244 Tester	17
16	S257 AltUAddr	LS283 NstackP	25S09 stackP	S260 ProcLow, IBEmptyErr	AM27S07 RH[4-7]	S374 IBFront	S373 IB[0]	S374 AlwaysClks	F93427 Kpc16Prom	16
15	S00 Byte',Nib' pMAR',X←0	25S09 SUAddrHigh	F93422 SU[0-3]	F93422 SU[4-7]	F93422 SU[8-11]	F93422 SU[12-15]	S374 @ AlwaysClks	F93453 CSIntProm		15
14	S138 fYNorm(Req)	25S09 SUAddrLow	S04 Mar←,RH←,c3, F#0,aD.0,IbE	S00 WrSU,WrLink, WrIBf,WrRH	LS32 @ Link, WrIB, ..., ...	S00 @ WrTC,C2Clk, Pop, ...	S253 R Ends	S38 Q Ends, CIN-SE	LS374 IOPTPCHigh	14
13	S138 fYNorm	S08 paSh0,MemEr pAllow,TCWt	S138 fX	S04 AlwysClk(3), WaitClk,C1,c2	S51 WriteTPC', Waitc1'	LS32 Link	S182 LookAhead	18Pin Plat Resistors	LS244 IOPTPCLow	13
12	q12 a IDM2901A-1 [0-3]	m12 b IDM2901A-1 [4-7]	c		n12 e IDM2901A-1 [8-11]	f	p12 g IDM2901A-1 [12-15]	h		12
11	S20 XBus←IB, LRotn	S138 fZNorm	S10 push,sh,Xbyte	S10 pTC.3,Wait, pTC.2	S151 DispBr.3A	S151 DispBr.2	S64 pNIA.8	S64 pNIA.9	S64 pNIA.10	11
10	S86 PgCr,Ref' Map,Ref	25S09 aSh,aFh	S374 rA, aS	S00 pTC.0,1, PgCross,CInpc	S151 DispBr.3B	LS158 DispBr.01	LS158 pNIA[4-7]	LS158 pNIA[0-3]	S64 pNIA.11	10
9	S374 fX, fZ	S175 CIn',fY.0,fS	S374 Misc,fY,fS	S374 rB,aD,aFl	AM27S07 Link	S374 TC, CSPar	AM27S07 TC	S374 NIAX[8-11]	25S09 NIA[8-11]	9
8	a LS251 IOPBus.0	b 93S48 pCSPar.0	c 93S48 pCSPar.1		e LS244 CSIn	f 93S48 pCSPar.2	g 93S48 pCSPar.3	h 18 ohm resistors		8
7	I2147L pfZ.0	LS251 IOPBus.1	LS251 IOPBus.2	LS251 IOPBus.3	LS251 IOPBus.4	LS251 IOPBus.5	LS251 IOPBus.6	LS251 IOPBus.7	25S09 NIA[0-3]	7
6	I2147L pfX.0	I2147L pfZ.1	I2147L pfZ.2	I2147L pfZ.3	I2147L INIA.8	I2147L INIA.9	I2147L INIA.10	I2147L INIA.11	25S09 NIA[4-7]	6
5	LS32 EnDispBr	I2147L pfX.1	I2147L pfX.2	I2147L pfX.3	I2147L INIA.4	I2147L INIA.5	I2147L INIA.6	I2147L INIA.7	PLAT logic analyzer	5
4	I2147L pfY.0	I2147L pfY.1	I2147L pfY.2	I2147L pfY.3	I2147L INIA.0	I2147L INIA.1	I2147L INIA.2	I2147L INIA.3	S374 NIAX[0-7]	4
3	I2147L pEP	I2147L pCIN-SE-WrSU	I2147L pEnSU	I2147L pmem	I2147L pfS.0	I2147L pfS.1	I2147L pfS.2	I2147L pfS.3	AM27S07 TPC[8-11]	3
2	I2147L paS.0	I2147L paS.1	I2147L paS.2	I2147L paF.0	I2147L paF.1	I2147L paF.2	I2147L paD.0	I2147L paD.1	AM27S07 TPC[0-3]	2
1	I2147L prA.0	I2147L prA.1	I2147L prA.2	I2147L prA.3	I2147L prB.0	I2147L prB.1	I2147L prB.2	I2147L prB.3	AM27S07 TPC[4-7]	1
	a	b	c	d	e	f	g	h	i	

6 holes must be drilled here & nets blue-wired

DIP Orient.

I/O Connector Area (Top) | I/O Connector Area (Bottom)

XEROX SDD	Project		File	Designer	Rev	Date	Page
	Dandelion	Stichweld Layout	LionHead52.sil	Garner	1	24 Mar 80	52



XEROX SDD	Project Dandelion	PC Layout	File LionHead53.sil	Designer Garner	Rev F	Date 2/?/78	page 53

I. Filter caps: 1 per 3 chip positions; 2 per 2147L; 1 per 2901.

II. Don't use PC layout positions a1, b1, a2, & b2.

III. There are 5 spare positions: c1, d1, c2, b3, 111.

IV. Clock qualifiers: The qualifier chips boxed in the PC layout should be kept together & near their current location (i.e. center of "board"). The S02 at pc loc a11 should not be moved.

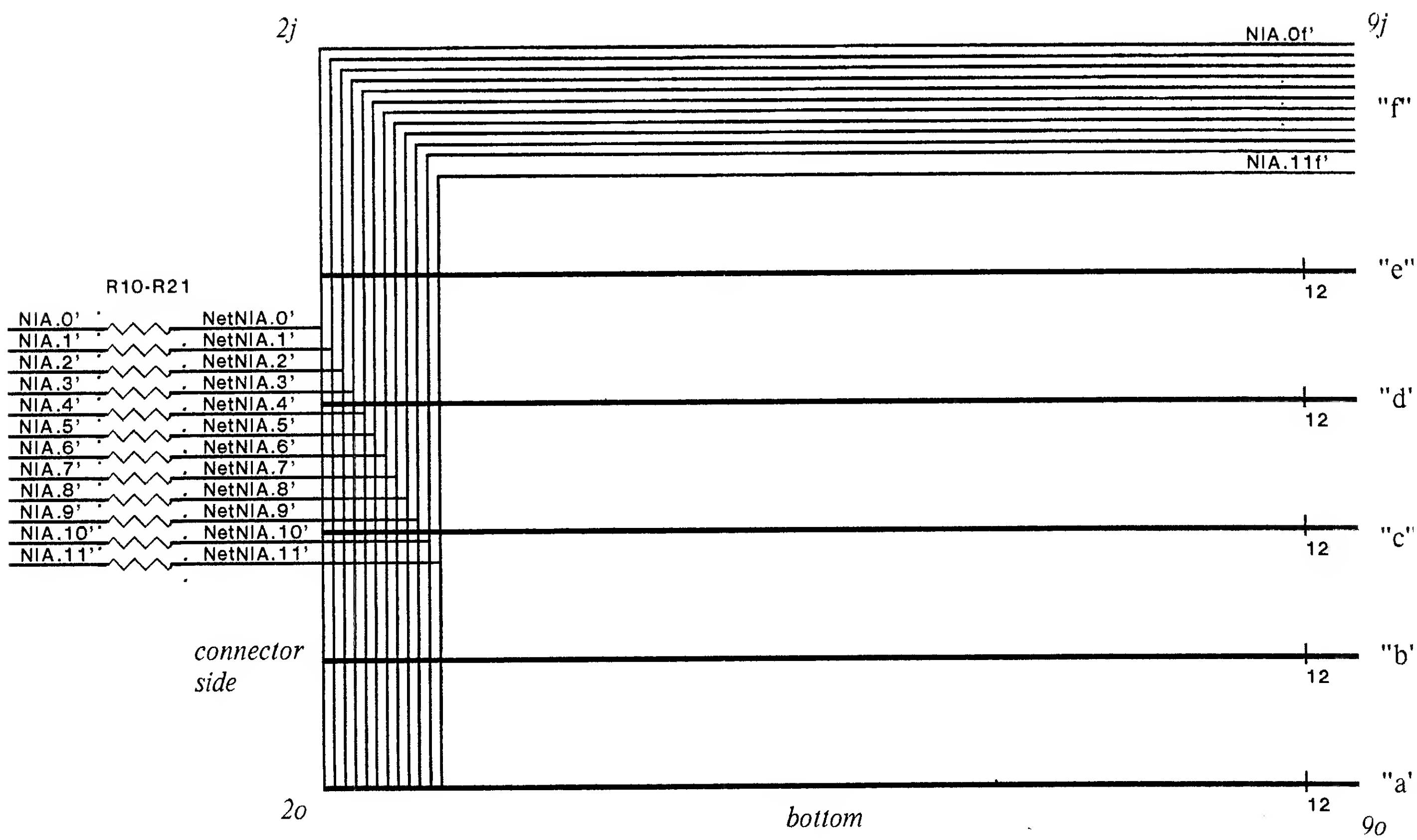
V. Control store layout

The CS is a 6 by 8 array with horizontal address lines & vertical data lines
The 8085 reads the CS from the bottom (via the LS251'), parity is computed at the bottom, and the MIR is located at the top.

Address Lines: Each horizontal row of 8 chips has its 12 address lines connected together- suffixed by "a" through "f" in the diagrams. The 8 rows are interconnected at the left side with a vertical bus, called NetNIA, which driven by the NIA register.

NetNIA is defined by the file NetNIA.sil.

R10-R21 are necessary to prevent undershooting & approximately equal the line impedance divided by 6. Electrolytic bypass caps may be necessary (2nd etch).



Calculated Delay:

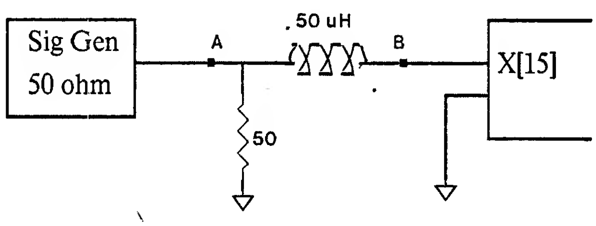
$$V = \frac{1}{C} \int i \; dt \;=\; \frac{t}{C} \left(\frac{i_{init} - i_{threshold}}{2} + i_{threshold} \right)$$

For standard Schottky, Fig. E3, p. 6-103 of the TI Data Book shows Iinit = 68 mA, Ithreshold = 42 mA. On page 49, the X bus capacitance is estimated at 327 pF. Therefore,

X bus delay = t = CV/I = (327 pF)(1.3 V)/(.055 A) = 8 nS

Measured Capacitance & Delay:

Using the following circuit, the capacitance of the X bus has been measured at 337 pF (for 2 PC-Dandelions @25 C)



Adjust frequency so that voltage @B is maximum.
Voltage @A is .4 Vpp centered above 2V.
All bus driver outputs are disabled.

$$C = \frac{1}{4\,(3.14)^2 f^2 L} = \frac{T^2}{1.97 \times 10^{3}}$$

Using a high-BW scope, the following delays were observed for X[15]:

w/ S240 drivers:	w/ S257 driver:	
7 nS	8 nS	On CP board (between driver and 2901 D input)
10 nS	11 nS	On backplane
16 nS	21 nS	On backplane w/ CP on card extender